特別研究報告

題目

仕様記述言語 JML と Alloy を用いた
在庫管理プログラムの設計に対するモデル検査ツールによる検証事例

指導教員
楠本 真二 教授

報告者
尾鷲方志

平成 19 年 2 月 20 日

大阪大学 基礎工学部 情報科学科

平成 20 年度 特別研究報告

仕様記述言語 JML と Alloy を用いた
在庫管理プログラムの設計に対するモデル検査ツールによる検証事例

尾鷲方志

内容梗概

ソフトウェアの開発において，従来から使われている手法として V 字型モデルによる設計法がある．この V 字型設計法において，近年，Design by Contract(契約に基づく設計，以下 DbC とする) による考え方が脚光を浴びている．DbC に基づいた開発では，アサーションにより契約を示す．アサーションを実現するための言語のとして Alloy や JML が挙げられる．Alloy は，Model Finder と呼ばれる AlloyAnalyzer を用いて，抽象度の高い設計において，仕様の矛盾発見などに役立つ．一方 JML は，Java コードと結びついており，抽象度の低い段階での設計，実装の検証に役立つ．このため，Alloy から JML に変換が可能ならば，抽象度が高く正当性のある程度保障された契約に対し，それ侵さぬよう低レベル設計が得られるため，より信頼性の高い開発を行うことができる．

本研究では，Alloy から JML 付きの Java のコードへの変換をすることを念頭において，そのための事例研究として，在庫管理プログラムの設計を Alloy と JML で行った．それぞれ AlloyAnalyzer と ESC/Java を用いてその正当性を検証した．また，各検証については，作成した在庫管理プログラムにおいて，警告内容について，なぜそのような警告がでたか，そして解決できたかについてまとめ考察した．結果，Alloy においては一部仕様を省略したが Alloy による述語の扱い方を工夫することにより JML と類似した記法により仕様記述をすることができた．

主な用語

JML，Alloy，契約による設計，設計検証，在庫管理問題

目 次

1　まえがき

ソフトウェアの開発手法において，従来から使われている手法としてV字型モデルによる設計法 [20] がある．V字型モデルによる設計法とは，開発における各フェーズを「設計プロセス」と「検証プロセス」に分割し，それぞれを検証内容によって対応させたものである．この設計において，全体の流れは抽象から詳述へ，といったようになっている．

V字型モデルによる設計法において，近年，Design by Contract(契約に基づく設計，以下DbCとする)[5][17] による考え方が脚光を浴びている．DbCに基づいた開発では，アサーションにより契約を示すことで，詳述段階におけるデバッグを効率化され，品質が向上する．この考え方はV字型モデルによる設計法に対して非常に有用である．

DbCを実現するための言語としてAlloy [15] やJML [16] が挙げられる．Alloyは，実行のためのコードの実態を持たず，宣言されたシグネチャに対し，適切な述語を書き，それを表明することで契約を表現する．また，AlloyAnalyzerを用いてその契約を満たすモデルのインスタンスや満たさない場合その反例を動的に探し出すことができ，抽象度の高い段階での設計に役立つ．一方JMLは，Javaコード実体と結びついた契約を記述することができ，ESC/Java [18][19] を用いて，コードが契約を侵す恐れがないかテストができるため，抽象度の低い段階での設計，実装に役立つ．

このため，AlloyからJML付きのJavaのコードに変換することができれば，抽象度の高い段階で仕様の正当性をテストされた仕様に対してコードがその契約を侵さぬようコーディングを行うことができるため，より信頼性の高い開発を行うことができる．

しかし，V字モデルによる開発におけるテスト段階に対して有用なJavaコードからAlloyへの変換に関する研究はJalloy [2] やTestEra [3] をはじめとし，さまざま見受けられるが，設計段階に対して有用な，AlloyからJML付きのJavaコードへの変換に関する研究は一部で言及 [14] されているものの，その成果物はあまり見られない．

本研究では，AlloyからJML付きJavaコードへの変換をすることを最終目的とし，そのために，具体的な事例に基づくケーススタディとして，在庫管理プログラム [11][12] の設計をAlloyとJMLそれぞれについて行った．JMLでの記述に関しては，ESC/Javaにおいて検証し，設計した仕様でプログラムが動作することを示すためにJavaによるプログラムの記述も行った．ESC/Javaを用いた検証で出力された警告に対し，どのような警告が出たか，また，作成した在庫管理プログラムのバージョンが上がるにつれ，どのように改善されたかをまとめ，考察した．また，AlloyAnalyzerを用いた検証に対して，アサーションが正当なものであることを示すだけでなく，Model Finder [15] としてのAlloyに踏み込んだ考察を行った．結果，JMLでESC/Javaにおいて正当と判断される契約を記述することができ，Alloyにおいては一部仕様を省略したが，Alloyによる述語の扱い方を工夫することにより

JML と類似した記法により仕様記述をすることができた.

2　準備

前章で述べたように，本研究では在庫管理プログラムの設計およびその設計の検証を行った．この章ではそのために必要な在庫管理プログラムの仕様について述べる．

2.1　在庫管理プログラムの仕様

在庫管理業務は図 1 に示されるように，受付係と倉庫係によって行われる．

それぞれがやり取りするデータとして，搬入単位であるコンテナとその内蔵品，取引先である顧客，その顧客からの要求が考えられる．

2.1.1　受付係の仕様

受付係は外部からの要求を受け取り，倉庫に対して在庫が要求を満たすか確認させ，可能ならばその要求に含まれる商品の出庫を倉庫係に依頼し，不可能であれば未処理要求リストとして保存する．同様に，外部からコンテナを受け付け，受け付けたコンテナを倉庫係に搬入させる．このとき，未処理の要求があればその要求の処理をする．

外部からアクセスできるのは受け付け係のみである．一つの受付係が担当する倉庫は一つのみであり，要素として倉庫係を持つ．

2.1.2　倉庫係の仕様

倉庫係は搬入したコンテナのリストをもっている．受付から送られてきた要求に対し，在庫が要求を満たしているかどうかをかえす．また，出庫指示に対して，どのコンテナからどれだけ出庫したかを示す出庫指示書を受付に返す．

全品目の集合の各要素は一意であり，また，コンテナの集合に対し，各コンテナのコンテナ番号は一意である．

品物の搬入はコンテナ単位で行われるが，顧客からの要求は品目単位で行われるため，コンテナの集合だけでなく，その中身をまとめた全品目の集合を要素として持つ．全品目の集合とコンテナの集合中の内蔵品の合計は互いに矛盾しない．

2.1.3　コンテナの仕様

コンテナは内蔵品の集合を持つ．各内蔵品は品目として表され，内蔵品の集合内の品目は一意である．また，倉庫からの出庫指示に対し，内蔵品の集合の中の品目を指定量取り出し，発送する．

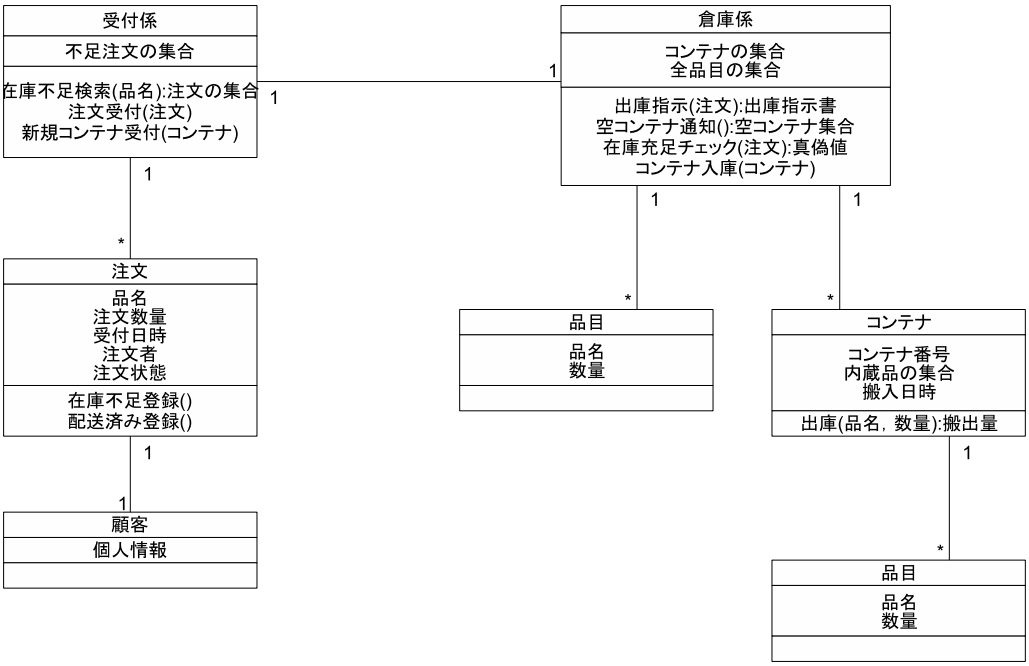

図 1: 在庫管理プログラムの仕様

2.1.4 品目の仕様

品目は要素として，品名，数量の情報を持つ.

2.1.5 要求の仕様

要求は，要素として，要求品名と要求数量，要求者 (顧客)，さらに要求が不足している，発送済みである等の状態を示す値を持つ．また，一度の要求に対して要求できる品目は一種類のみである.

2.1.6 顧客の仕様

顧客は，顧客の住所，氏名などの個人情報を持つ.

3 JML による設計

3.1 JML による契約

Java コード中に JML を記述する際，特殊なコメントの形で記述する．このコメントは通常の Java のコメント開始記号に@記号が続く形で，具体的には`/*@ 〜 */`や，`//@ 〜` 等の形で記述される．

JML において，契約は主に以下のようなもので示される．

事前条件 (requires) 各メソッドの処理に入る前に満たされているべき条件を示す.

事後条件 (ensures) 各メソッドの処理が終了した時に満たされているべき条件を示す.

不変条件 (invariant) この定義を持つオブジェクトが常に満たしているべき条件を示す.

代入可否 (assignable) 各メソッドにおいてどのメンバ変数に対する代入を許可するかを示す.

以降，前章において示された仕様を満たすこれらの条件について考察した設計を示す.

3.2 必要としたクラス

前章の仕様で示された各部をそれぞれクラスで表現する．対応は表 1 の通りである．このように分けられたクラスに対して示した契約についてそれぞれ述べる.

受付係	ReceiptionDesk
倉庫係	Storage
コンテナ	ContainerItem
品目	Item
要求	Request
顧客	Customer

表 1: 仕様とクラスの対応

また，倉庫係が出庫指示の際に受付係に返す出庫指示書を示すのに必要なデータを考える.

一度の要求に対して出庫される品目は一種類である．また，複数のコンテナから取り出されるから，どのコンテナからどれだけ取り出したかを記録しておきたい.

このため，内蔵品を一つしか持たないコンテナのリストとして出庫指示書を表現することにする.

以後各クラスとそのメソッドが持つ契約について述べるが，getter や setter 等の単純な操作や初期値を与えるだけのコンストラクタについての契約は省略する．

3.3 ReceiptionDesk クラス

3.3.1 不変条件

ReceiptionDesk クラスは要素として倉庫と未処理要求のリストを持っている．これらは空でもかまわないが実体を持っている必要があるため null にはならない．また，未処理要求のリストに入っているものは要求，つまり Request でなければならない．これを示したのが図 2 である．ここで用いられている\typeof() は括弧内で指定された変数の持つオブジェクトの型を表す．また，\type() はカッコ内で示されたデータ型の型情報を表す．

```
private /*@ spec_public non_null @*/ List RequestList;
private /*@ spec_public non_null @*/ Storage storage;

/*@ public invariant \typeof(RequestList) == \type(Request); @*/
```

図 2: クラス ReceiptionDesk の不変条件

3.3.2 メソッド receiptRequest(Request r)

ReceiptionDesk のメソッド receiptRequest は引数として要求を取り，その要求を満たす在庫があるならば出庫指示を出し，そうでなければ未処理要求リストにこの要求を追加する．このメソッドに対する契約を図 3 に示す．

まず，public behavior として記述されていることから，この契約はこのメソッドの一般的な振る舞いについて示していることを表す．

事前条件

```
/*@ public behavior
    requires r != null;
    assignable storage, RequestList;
    ensures \result == storage.checkStockSatisfied(r);
 @*/
```

図 3: ReceiptionDesk のメソッド recieptRequest(Request r) に対する契約

図 3 に示したように，requires 節で事前条件が定められる．引数として受け取る要求の値

を参照するために引数は null ではならない．後述の Request クラスにおいて，コンストラクタにて適切な値で初期化されることが保障されているので引数 r に対してこれ以上の事前条件を決定しておく必要はない．

事後条件　このメソッドは要求が満たされている場合は true，満たされずに未処理要求リストに要求を追加した場合に false をかえす．したがって戻り値は，図 3 の ensures 節で示されたように，storage のメソッド checkStockSatisfied が真ならば真でなければならない．ここで，\result は戻り値を表している．

メソッド checkStockSatisfied については Storage クラスの節において述べる．

代入可否　図 3 に示したように，assignable 節で定められる．このメソッドにおいて，引数で与えられた要求を在庫が満たしているならば出庫処理が行われるので倉庫係，この場合，要素 storage が変更される可能性がある．また，要求が満たされていない場合は未処理要求リストである RequestList が更新される必要がある．したがってこの二つの要素を assignable 節で表明しておく．

3.3.3　メソッド deliveringOrder()

ReceiptionDesk のメソッド deliveringOrder はすべての未処理要求リストに対して出庫指示を試みる．それぞれの要求に対し，在庫が充足しているならば出庫指示が実際に行われ，要求の状態は配送済みになり，そうでなければ未配送のままである．

また，このメソッドは各要求に対する出庫指示書のリストを返す．

このメソッドに対する契約を図 4 に示す．

事前条件　このメソッドは引数を持たず，そのほかの参照する値については不変条件で有効であることが表明されているので，特にここで宣言すべき事前条件を持たない．

事後条件　事後条件は，未処理要求リストに対し，それぞれの要求がどのような状態になったか保障しなければならない．まず，条件を満たしている要求について考える．未処理要求リスト中の任意の要求に対し，在庫が要求を満たしているならば，storage から出庫した際に戻ってくる出庫指示書により示される，各コンテナから要求品目をどれだけ出庫したかについて，その合計値が要求量に等しいはずである．また，この在庫管理プログラムは，未処理要求リストから処理済の要求を削除する処理を持たないため，ここで配送されたもの以外にも配送済み状態の要求がいくつか存在する．したがって，配送済みならば上記が成立するのではなく，上記が成立するならば，配送済みと言うことができる．図 4 で示される最初の

```
/* public behavior
        assignable RequestList,storage;
        ensures (\forall Request r; RequestList.contains(r)
                && (\sum ContainerItem ci;\old(storage).deliveringOrder(r).contains(ci);
                        ((Item)ci.getContainedItem().get(0)).getAmount())==r.getAmount();
                r.getRequestState() == StockState.DELIVERED);
        ensures (\forall Request r; RequestList.contains(r)
                && r.getRequestState() == StockState.SHORTAGE;
                \old(storage).checkStockSatisfied(r)==false);
        ensures (\forall Request r;RequestList.contains(r)
                && r.getRequestState()==StockState.DELIVERED
                && \old(r.getRequestState())==StockState.SHORTAGE;
                \result.contains(\old(storage).deliveringOrder(r)));
  */
```

図 4: ReceiptionDesk のメソッド deliveringOrder() に対する契約

ensures 節はこれを示す．ここで用いられている構文について簡単に述べる．

$$(\backslash forall \text{ 変数宣言}; boolean \text{ 式 } 1; boolean \text{ 式 } 2) \tag{1}$$

この構文は，boolean 式 1 を満たす任意の変数に対し，boolea 式 2 が成立することを示し，

$$(\backslash sum \text{ 変数宣言}; boolean \text{ 式}; \text{式}) \tag{2}$$

この構文は，boolean 式を満たす任意の変数に対し，式により示された値の和をあらわす．

また，\old(〜) は，括弧中の変数をこのメソッド実行前のものとして扱うための構文である．

次に，このメソッドを終えて未配送状態の要求について考える．未配送のものは必ず storage の checkStockSatisfied を通した際に偽であるから二番目の ensures 節にはこれを記述しておく．

最後に戻り値について考える．各要求に対し出庫指示書が発行される．戻り値はそれらをすべて含んでいる必要がある．それらの要求は，処理前には不足状態であり，処理後に配送済み状態であるというように状態が変化している．この ensure 節はこれが成立する要求に対する出庫指示書に対し，戻り値\result がそれを含むことを保障している．

代入可否　このメソッドは，未処理の要求リストを処理する際，在庫が要求を満たしているならば発送済みであると，要求の保持している状態を書き換えるので，RequestList は assignable である．また，出庫処理を行う際に，倉庫中のコンテナから品物を取り出すので，storage も assignable である．したがって，図 4 で示したようになる．

3.3.4 メソッド receiptNewContainerItem(ContainerItem c)

このメソッドは，受け付けたコンテナを倉庫係に渡し，さらに，在庫が変動したことにより満たされる要求があるならそれについて出庫指示を行う．このメソッドに対する契約は図4に示される．

```
/* public behavior
        requires c != null;
        assignable storage;
        ensures (\exists ContainerItem ci;
                 storage.getContainerItemList().contains(ci);
                 ci.getContainerID()==c.getContainerID());
  */
```

図 5: ReceiptionDesk のメソッド receiptNewContainerItem(ContainerItem c) に対する契約

事前条件　図5の requires 節で示されるように，引数であるコンテナが有効であることのみである．c が null でなければ ContainerItem クラスのコンストラクタにより適切な値を持つことが保障されている．

事後条件　コンテナを倉庫に登録するため，このコンテナが倉庫のコンテナリストに含まれる．しかし，未発送の要求を処理する際に登録したコンテナの中身が発送される可能性があるため，厳密に同じコンテナが含まれることは保障できない．しかし，仕様により倉庫係のコンテナリストに含まれるコンテナ番号は一意であるため，引数のコンテナ番号を一致するコンテナ番号を持つコンテナが倉庫のコンテナリストに含まれることは保障される．図5で示す ensures 節は上記のことを示す．ここで用いられている構文\exists について述べる．

$$(\backslash exists \text{ 変数宣言}; boolean \text{ 式 } 1; boolean \text{ 式 } 2) \tag{3}$$

この構文は，boolean 式 1 を満たすある変数に対し，boolea 式 2 が成立するものが存在することを表す．

代入可否　このメソッドでは倉庫係にコンテナを登録するために storage に対する値の操作が発生する．したがって assignable 節には storage が含まれる．

3.4 Storage クラス

3.4.1 不変条件

Storage クラスは要素として登録されたコンテナを表す containerlist と，コンテナの全内蔵品を品目ごとに保持する allitemlist を持つ．これらは空でもよいが，null ではならないため non_null として宣言する．また，コンテナリストの要素はコンテナでなくてはならず，品目リストの要素は品目でなければならない．これを示すために前述の\tyoepf(変数)と\type(型) を用いた．

さらに，仕様によりコンテナリスト内のコンテナ番号は一意である．同様に品目リストの品名も一意である．これらについては前述の構文 1 を用いてコンテナ内の異なる任意のコンテナは番号が違うこと，全品目リスト内の異なる任意の品目は名前が異なることを表した．

これらのことを図 6 に示す．

```
private /* spec_public non_null */ LinkedList containerlist = new LinkedList();
private /* spec_public non_null */ LinkedList allitemlist = new LinkedList();

/*      public invariant \typeof(containerlist) == \type(ContainerItem);
        public invariant \typeof(allitemlist) == \type(Item);
        public invariant (\forall Item i,j;allitemlist.contains(i) && allitemlist.contains(j)
            && i!=j;!i.getName().equals(j.getName()));
        public invariant (\forall ContainerItem ci, cj;
            containerlist.contains(ci) && containerlist.contains(cj) && ci!=cj;
            ci.getContainerID()!=cj.getContainerID());
  */
```

図 6: クラス Storage の不変条件

3.4.2 メソッド registContainerItem(ContainerItem c)

このメソッドは引数により渡されたコンテナをコンテナリストに登録し，全品目リストを更新する．このメソッドに対する契約は図 7 に示す．

```
/* public behavior
        requires c != null;
        requires (\forall ContainerItem i; containerlist.contains(i);
                i.getContainerID()!=c.getContainerID());
        requires (\forall Item i,j; c.getContainedItem().contains(i)
                && c.getContainedItem().contains(j);!i.getName().equals(j.getName()));
        assignable containerlist, allitemlist;
        ensures containerlist.contains(c);
        ensures (\forall ContainerItem i;\old(containerlist).contains(i);
                i.getContainerID()!=c.getContainerID());
        ensures (\forall Item i,j; c.getContainedItem().contains(i)
                && allitemlist.contains(j) && \old(j.getName().equals(i.getName()));
                        j.getAmount() == \old(j.getAmount()+i.getAmount())
        );
        ensures (\forall Item i,j; c.getContainedItem().contains(i)
                && allitemlist.contains(j) && !\old(allitemlist).contains(j);
                j.getAmount() == i.getAmount()
        );
        ensures !\old(allitemlist.isEmpty()) ==> !allitemlist.equals(c.getContainedItem());
  */
```

図 7: Storage クラスの registContainerItem(ContainerItem c) に対する契約

事前条件　まず，引数で渡されるコンテナ c が有効でなければならない．コンテナ c が初期化されているならば，コンストラクタにより有効な値が入っていることが後述のクラス ContainerItem にかかれた契約により保障されている．

また，引数で渡されたコンテナが空ではないとして，コンテナリスト内のコンテナ番号は一意でなければならないため，倉庫内の containeritemlist に存在するコンテナ番号と c のコンテナ番号が同じものが存在してはならない．また，c は同じ種類の内蔵品を複数に分けて持っていてはならない．これらのことを前述の構文 1 を用いて，図 6 の require 節にて表した．

事後条件　コンテナを登録するので，搬入されたコンテナがコンテナリストに含まれなければならない．これを最初の ensures 節で表した．次の ensures 節では，搬入されたコンテナ ID と同じものが搬入前には存在しないことを\old() を用いて表すことにより，搬入後のコンテナ番号の一意性を保障した．3 番目の ensures 節では，コンテナが入庫することにより更新される全品目リストについて，搬入されたコンテナの中身と同じ品目については在庫が搬入された分だけ増えることを表した．4 番目の ensures 節にて，このメソッド実行前には

全品目に含まれず，終了後にコンテナが搬入されることにより含まれるようになった品目については，搬入されたものと同じ量だけ在庫に入っていることを表した．最後に，搬入前に在庫が空でないならば，全品目リストが搬入したコンテナの中身と完全に同じになることはない事を表した．

代入可否　このメソッドにおいては全品目リストである allitemlist とコンテナリストである containerlist が更新されるので，これらを assignable として契約を表した．

3.4.3　メソッド updateAllItemList(ContainerItem c)

このメソッドは前述の registContainerItem から呼び出され，搬入されたコンテナの中身をもとに全品目リストを更新する．契約は図 8 のように表した．registContainerItem から Storage クラス内の containerlist に関する契約を省いたもので，それらを除けばまったく同じ契約が表されている．そのためここでは解説を省略する．

```
/* public behavior
        requires c != null;
        requires (\forall Item i,j; c.getContainedItem().contains(i)
                && c.getContainedItem().contains(j);
                !i.getName().equals(j.getName()));
        assignable allitemlist;
        ensures (\forall Item i,j; c.getContainedItem().contains(i)
                && allitemlist.contains(j) && \old(j.getName().equals(i.getName()));
                 j.getAmount() == \old(j.getAmount()+i.getAmount())
        );                                                                          10
        ensures (\forall Item i,j; c.getContainedItem().contains(i) && allitemlist.contains(j)
                && !\old(allitemlist).contains(j) && j.getName().equals(i.getName());
                j.getAmount() == i.getAmount()
        );
        ensures !\old(allitemlist.isEmpty()) ==> !allitemlist.equals(c.getContainedItem());
  */
```

図 8: Storage クラスの updateAllItemList(ContainerItem c) に対する契約

3.4.4　メソッド informEmptyContainer()

このメソッドは倉庫の中のコンテナの内，全品目の残量が空のコンテナのリストを返す．契約は図 9 のように表される．

```
/* public behavior
        ensures containerlist.containsAll(\result);
        ensures (\forall ContainerItem ci; \result.contains(ci);
                (\forall Item i; ci.getContainedItem().contains(i); i.getAmount()==0)
        );
        assignable \nothing;
  */
```

図 9: Storage クラスのメソッド informEmptyContainer() に対する契約

事前条件　引数を持たず，また，内部の参照する変数に関しても null ではないことが既に表明されているので特に示すべき事前条件はない.

事後条件　戻り値として返すべきものは Storage クラス内のコンテナリスト中の空のコンテナのリストであるから，コンテナリストはこれらをすべて含んでいなければならない．これを最初の ensures 節に示す.

また，戻り値に含まれるコンテナの中身のすべての品目が 0 であることを二重の forall 構文 1 を用いて表す.

代入可否　このメソッドにおいて変更される値が存在しないので，\nothing で存在しないことを表す.

3.4.5　メソッド checkStockSatisfied(Request r)

このメソッドは引数として注文が与えられたとき，その注文を在庫が満たしているかを調べる．このメソッドに対する契約は図 10 に示す.

```
/* public behavior
        requires r != null;
        assignable \nothing;
        ensures \result == (\exists Item i;allitemlist.contains(i);
                        r.getName().equals(i.getName())
                && r.getAmount() <=i.getAmount());
  */
```

図 10: Storage クラスの checkStockSatisfied(Request r) に対する契約

事前条件　このメソッドは r で表される注文が空ではないことを要求する．項 3.3.2 にて述べたように，r が空でないならば適切な値が入っていることは後述の Request クラスにて保障されている．

事後条件　前述の exists 構文 3 を用いて，要求品目と同じ名前かつ，要求量以上の在庫が存在するならばそのときに限り戻り値に true を返すことを保障している．

代入可否　このメソッドにて Storage クラスのフィールドの変数の値の変更が発生しないので，\nothing で存在しないことを表す．

3.4.6　メソッド deliveringOrder(Request r)

このメソッドは受付 ReceiptionDesk から受け取った注文をコンテナに対して出庫指示を出し，呼び出し元である受け付けに対し出庫指示書を返す．図 11 に契約を示す．

```
/* public behavior
        requires r != null;
        requires checkStockSatisfied(r);
        assignable allitemlist, containerlist;
        ensures (\sum ContainerItem i; \result.contains(i);
                ((Item)i.getContainedItem().get(0)).getAmount()) == r.getAmount();
        ensures (\forall ContainerItem i; \result.contains(i);
                ((Item)i.getContainedItem().get(0)).getName().equals(r.getName()));
  */
```

図 11: Storage クラスの deliveringOrder(Request r) に対する契約

事前条件　このメソッドは r で表される注文が空ではないことを要求する．項 3.3.2 にて述べたように，r が空でないならば適切な値が入っていることは後述の Request クラスにて保障されている．また，仕様より，出庫をするには在庫が足りていなければならないので，次の ensures 節では，前述の checkStockSatisfied を用いて注文が出庫可能であることを要求している．

事後条件　まず，適切に出庫されたならば，出庫指示書で示される各コンテナから出庫された品目の数量の合計は注文された量と等しいはずである．これを，前述の sum 構文 2 を用いて最初の ensures 節のように表した．

次に，出庫指示書に記されている出庫された品目は注文した品物と同じでなければならない．これを次の ensures 節にて示した．

代入可否　この操作にて，コンテナリストと全品目リストの数量がそれぞれ変更されるため，assignable な変数として containerlist と allitemlist を表明した．

3.5　ContainerItem クラス

3.5.1　不変条件

このクラスは搬入されるコンテナを表す．フィールド内に，搬入日時を表す carryingDate と内蔵品を表す品目のリスト itemlist を持ち，それらは null ではならない．また，itemlist の持つ要素は品目である Item 型でなければならず，itemlist 中に含まれるすべての異なる品目は名前が一意でなければならない．

以上のことを図 12 にて表した．

```
private /* spec_public non_null */ Date carryingDate;
private int containerID;
private /* spec_public non_null */ LinkedList itemlist;
/* public invariant containerID >= 0;
        public invariant \typeof(itemlist) == \type(Item);
        public invariant (\forall Item i,j;
                          itemlist.contains(i) && itemlist.contains(j)
                          && i != j;
                          i.getName() !=j.getName());
  */
```

図 12: クラス ContainerItem の不変条件

3.5.2　メソッド shippingItem(String name, int num)

倉庫係からの出庫指示を受けて引数にて指定された名前の品目を指定数量分コンテナから取り出す．戻り値として要求量に対する不足量を返す．一つのコンテナで要求を満足するときは負の数が帰ることもある．図 13 にて契約を示す．

```
/* public behavior
        requires name != null && !name.equals("");
        requires num > 0;
        ensures (\exists Item i; itemlist.contains(i);
                i.getName().equals(name)&&\result == \old(num-i.getAmount()));
        assignable itemlist;
  */
```

図 13: メソッド shippingItem(String name, int num)

事前条件　まず，最初の ensures 節は，引数で指定される name が null ではなく，有効な名前，ここでは一文字も持たない名前であってはならないことを示す．また，次の ensures 節では，引き出す品目の数は正の数であることを要求している．

事後条件　exists 構文 3 を用いて，コンテナの内蔵品リストである itemlist には name と同じ品名を持つ品目があり，かつ戻り値が要求量である num からその品目のメソッド実行前時点での個数を引いたものと等しくなる品目が存在することを示す．

代入可否　内蔵品リストに対する値の操作が発生するので，assignable な変数は itemlist である．

3.6　Item クラス

3.6.1　不変条件

このクラスは品目を表す．このフィールド中の変数に関して，品名を表す変数 name は null ではない．また，数量を表す totalamount は getter を通じて外部から参照され得る．

totalamount は負であってはならず，また，name には有効な名前が示されていなければならない．といった契約が考えられる．

これを図 14 に示した．

```
private /* spec_public non_null */ String name;
private /* spec_public */ int totalamount;

/* public invariant totalamount >= 0
                && name != null && !name.equals("");
  */
```

図 14: クラス Item の不変条件

3.7 Request クラス

3.7.1 不変条件

このクラスは注文を表す．receiptionDate は注文を受け付けた日付なので，注文が発生した瞬間に発生するから null であってはならない．配送日付を表す deliverlingDate は配送されるまで有効な値が入ることはないので特に条件を設けない．要求数量を表す amount は getter で外部に対し公開されるので public とした．

また，注文にとって注文者が存在しないはずがないので注文者も null であってはならないとした．

また，要求数量は必ず正である．

これらの契約を図 15 に示した．

```
private /* spec_public non_null */
        Date receiptionDate;
private
        Date deliveringDate;
private /* spec_public non_null */
        String itemName;
private /* spec_public */
        int amount;
private
        byte requestState;
private /* spec_public non_null */
        Customer customer;

/* public invariant amount>0; */
```

図 15: クラス Request の不変条件

3.8 Customer クラス

3.8.1 不変条件

顧客を表す Customer クラスは氏名，住所，郵便番号を示す要素 name,address,zipcode をフィールドに持つ．これらはすべて有効でなければならないし，外部から getter で参照するので public で null でないことが不変条件として挙げられる．

また，これらが文字を一つも持たない文字列であってはならないので，空の文字列と等しくないことを不変条件として表す．

これらの契約を図 16 に表した．

```
private /* spec_public non_null */
        String name;
private /* spec_public non_null */
        String address;
private /* spec_public non_null */
        String zipcode;

/* public invariant !name.equals("");
        public invariant !address.equals("");
        public invariant !zipcode.equals("");
  */
```

図 16: クラス Customer の不変条件

3.9 JML 記述のまとめ

以上のように，仕様で定められた全ての機能に対し，JML を用いて設計をすることができた．

この設計の過程において，表 2 に示されるような版を経て，最終的に表 3 に示される規模になった．

Version	
第一版	コンテナの内蔵品 (ContainedItem) と品目 (Item) が別クラスだった
第二版	上記クラスを Item に統合
第三版	設計に加えて Java のコードも完成，事後条件の修正
第四版	仕様を誤って実装していた点を修正

表 2: JML 設計改定履歴

	LOC	JML の LOC
ReceiptionDesk	141	55
Storage	242	97
ContainerItem	171	69
Request	185	87
Item	78	37
Customer	122	55
総計	939	380

表 3: JML 設計の規模

4 ESC/Java による検証

設計が正当かどうかを示すには，要求仕様で与えられた入力出力それぞれに関して行われる操作において，ESC/Java にとって正当であるということができればよいと考えた．以下，入出力それぞれに関わる操作について検証した事例を述べる．

4.1 入力操作について

設計した在庫管理プログラムは外部からの入力として，注文と新規コンテナが挙げられる．

4.1.1 コンテナ受付

入力として与えられたコンテナは，まず，ReceiptionDesk クラスのメソッド，receiptNewContainerItem メソッドで受け取る．このメソッドでは，内部で Storage クラスの registContainerItem メソッドを呼び出す．ここで，コンテナが搬入されたことにより品目リストを更新する動作が発生するため，Storage クラスの updateAllItemList メソッドが呼び出される．コンテナを受け付け，在庫が更新されたことにより未処理注文リストの確認と可能ならば出庫という操作が発生する．これは ReceiptionDesk の DeliveringOrder メソッドによって行われるが，DeliveringOrder メソッドについては出力項で検証する．

これら発生したそれぞれの操作に対し設計と実装が ESC/Java 的に正当であるということができるならば，コンテナ受付操作が ESC/Java 的に正当であるという検証結果が得られた，ということができる．

以下にそれぞれの操作に対する出力を示す．

```
StockManagement.ReceiptionDesk:
        receiptNewContainerItem(StockManagement.ContainerItem) ...
    [5.063 s 27627512 bytes]  passed

StockManagement.Storage:
        registContainerItem(StockManagement.ContainerItem) ...
    [3.75 s 19131536 bytes]  passed

StockManagement.Storage:
        updateAllItemList(StockManagement.ContainerItem) ...
Caution: Unable to check method
        updateAllItemList(StockManagement.ContainerItem) of type
```

```
                StockManagement.Storage because its VC is too large
    [0.797 s 25322192 bytes]  VC too big
```

receiptNewContainerItem と registContainerItem に関しては ESC/Java は異常無しと判断し，passed の出力を返した．

一方，updateAllItemList については，VC too big と返している．VC とは Verification Conditions の略で，これが大きすぎて JML ではなく Java によるプログラムコードと照らし合わせた検証ができないことを示す．

一方，あるメソッドを呼び出すメソッドでの契約を検証する際は，呼び出されたメソッドに対する契約をもとに検証される．したがって，updateAllItemList を呼び出すメソッドにおいて正当であるという検証結果が得られたため，JML 記述に関していうならば異常無しと言うことができる．

したがって，JML において ESC/Java 的に正当な設計であると，コンテナ受付操作に関しては言うことができた．

4.1.2 注文受付

注文受付では，まず，ReceiptionDesk の receiptRequest メソッドで注文を受け取る．受け取った注文が発送可能か，Storage クラスの checkStockSatisfied メソッドで確認し，可能ならば Storage クラスの deliveringOrder メソッドで発送し，そうでないならば未処理要求リストに加える．これらの操作について正当ならば注文の受付が政党に行われているということができる．Storage クラスのの deliveringOrder メソッドについても出力の節にて検証する．

以下に検証結果を示す．

```
StockManagement.ReceiptionDesk:
        receiptRequest(StockManagement.Request) ...
    [6.0 s 21911496 bytes]  passed


StockManagement.Storage:
        checkStockSatisfied(StockManagement.Request) ...
    [3.078 s 19392984 bytes]  passed
```

これらのメソッドに関しては，すべての操作に対し ESC/Java において正当であることが示されたので，注文受付操作については JML 的に正当であるということができ，実装も正当に行われていると言える．

4.2 出力操作について

4.2.1 出庫指示書

出庫指示書は Storage クラスの deliveringOrder にて出力される．また，未処理注文リストに関しては，ReceiptionDesk クラスにおいて，一括した出庫指示書のリストとしてこちらのクラスの deliveringOrder にて出力を得られる．ReceiptionDesk の deliveringOrder からは個々の注文に対し，Storage クラスの checkStockSatisfied と deliveringOrder が呼び出される．

Storage クラスの deliveringOrder からは，ContainerItem クラスのメソッドである shippingItem が呼び出され，内蔵品の setter を用いて値を変更し，出庫操作を表している．

以下にこれらのメソッドに対する検証結果を示す．

```
StockManagement.ReceiptionDesk: deliveringOrder() ...
    [6.64 s 24705360 bytes]  passed

StockManagement.Storage:
        deliveringOrder(StockManagement.Request) ...
Caution: Unable to check method
        deliveringOrder(StockManagement.Request) of type
            StockManagement.Storage because its VC is too large
    [1.328 s 30618536 bytes]  VC too big

StockManagement.ContainerItem:
        shippingItem(java.lang.String, int) ...
    [19.0 s 21894248 bytes]  passed

StockManagement.Item: setAmount(int) ...
    [0.219 s 13973096 bytes]  passed
```

これらの操作に対し，VC が大きすぎて Java コードと照らし合わせた検証が不可能な Storage クラスの deliveringOrder を除き，全ての操作が ESC/Java において正当であることが検証された．また，検証ができなかった Storage クラスの deliveringOrder を利用する ReceiptionDesk の deliveringOrder においては正当であるという結果から，storage クラスの deliveringOrder に関して，JML 記述においては正当な契約を示しているということができる．

したがって，ESC/Java 的に出庫指示書の出力に対する JML 記述による設計は正当である，と言える．

4.3 その他の動作

4.3.1 在庫更新時の出庫処理

前述のコンテナ受付処理において，ReceiptionDesk の deliveringOrder により出庫指示操作が行われている．ここで，各々の注文に対し，checkStockSatisfied で在庫充足がチェックされ，それに対し Storage クラスの deliveringOrder で出庫指示をコンテナに出し，shippingItem で出庫が行われる．

これらのメソッドについてはそれぞれ JML 記述が ESC/Java 的に正当であることが言えるので，新規コンテナ受付による在庫更新時に行われている出庫処理が正当であると言うことができる．

4.4 検証にかかった時間

表 4 に検証にかかった時間をまとめた．なお，検証環境は表 5 のようになっている．

クラス	メソッド	時間 (s)
ReceiptionDesk		22.546
	receiptRequest	6.0
	deliveringOrder	6.64
	getRequestList	0.031
	getStorage	0.047
	getItemList	0.313
	getContainerItemList	0.187
	receiptNewContainerItem	5.063
	ReceiptionDesk	0.75
Storage		20.609
	registContainerItem	3.75
	updateAllItemList	0.797
	informEmptyContainer	5.672
	checkStockSatisfied	3.078
	deliveringOrder	1.328
	getItemList	0.25
	getContainerItemList	0.156

Storage	2.766
ContainerItem	44.149
shippingItem	19.0
getContainerID	0.078
getAmount	8.11
getReceiptionDate	0.062
getContainedItem	0.282
ContainerItem(int, List, Date)	2.281
ContainerItem(int, List)	2.297
ContainerItem(iint)	1.672
toString	6.656
Request	10.281
registShortageStock	0.14
registAlreadyDelivered	0.063
getCustomer	0.015
setAmount	0.078
getAmount	0.032
getRequestState	0.015
getReceiptionDate	0.063
compareTo	0.359
getName	0.25
Request(String, int, Customer, Date, byte)	2.844
Request(String, int, Customer, Date)	2.812
Request(String, int, Customer)	1.125
Item	23.375
getName	0.688
getAmount	0.281
setAmount	0.219
toString	18.894
Item	1.0
Customer	18.219
informShortageStock	0.219
getName	0.36

getAddress	0.39
getZipCode	0.297
equals	2.891
setName	1.094
setAddress	1.174
setZipCode	1.204
Customer(String, String, String)	5.203
Customer()	3.125
総計	193.179

表 4: ESC/Java による検証にかかった時間

CPU	Pentium4 2.66GHz
RAM	1GByte
OS	WindowsXP Professional SP2
JDK	1.4.2
ESC/Java	2.0b

表 5: 検証環境

5　Alloy による設計

以下に Alloy を用いた契約の記述について示す.

JML と比較して，Alloy による契約の記述の仕方は考え方が大きく異なる．このため，最初に Alloy を用いたの契約の記述の仕方について述べる.

また，受付係などの一部クラスと，直接仕様を満たすのに必要のないメソッドは省略したため，JML による設計の解説とは違い，各々のオブジェクトに関しては不変条件についてのみ述べた上で，在庫管理プログラムに必要な機能である個々の動作に対応するメソッドについて述べる.

5.1　Alloy による契約

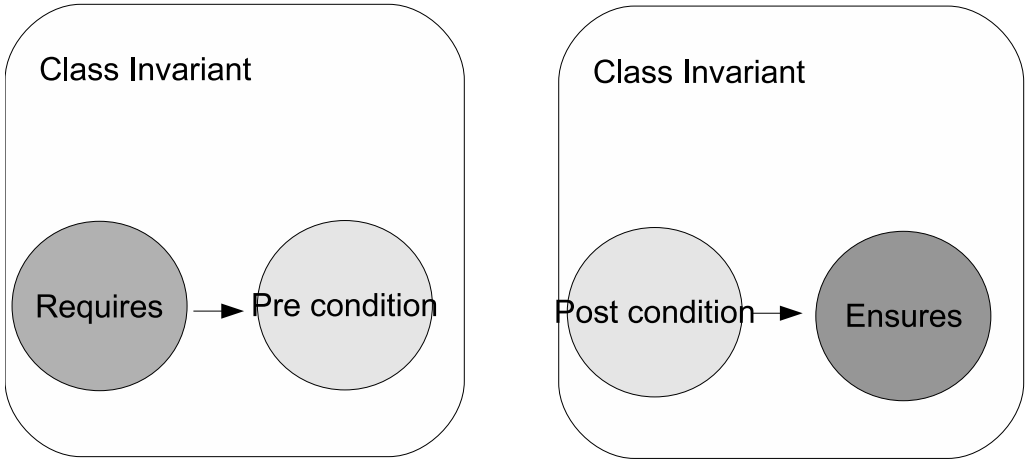

図 17: Alloy によるメソッドに対する契約

Alloy を用いて各メソッドにおける契約を示すことを考える.

図 17 によって示されるように，クラスの不変条件の中で事前条件として要求されるものが成立する状態において成立するモデルについて示す述語を考える (モデル A とする)．次に，同じくクラスの不変条件の中で，あるモデルに対し，事後条件と同じ内容で指定された述語が成り立つモデルを示す述語を考える (モデル B とする).

このモデル A があるメソッドの処理前の状態を表し，モデル B があるメソッドの処理後の状態を表す．モデル A，B は独立に成立するが，それぞれに同じ引数を与えた場合に，共に成立するモデルが存在するならば，それがメソッドの処理を示すモデルの一つである．

上記のような考え方で各クラスのメソッドのモデルを記述した．

5.2 各クラスの不変式

5.2.1 Storage クラス

```
fact Storage_Invariant{
        all s:Storage,c:ContainerItem|
                c in s.containerlist => #c.itemlist > 0
        all s:Storage|
                #s.containerlist > 0 => #s.allitemlist>0
        all s:Storage|
                #s.allitemlist > 0 => #s.containerlist>0
        all s:Storage|
                #s.containerlist = 0 => #s.allitemlist=0
        all s:Storage|                                                  10
                #s.allitemlist=0 => #s.containerlist=0
        all s:Storage,i,j:Item|
                i in s.allitemlist && j in (s.allitemlist-i)
                        => i.getName[] != j.getName[] && #(s.allitemlist & i) = 1
        all s:Storage,i,j:ContainerItem|
                i in s.containerlist && j in (s.containerlist-i)
                        => i.getContainerID[]!=j.getContainerID[]
        all s:Storage|
                s.containerlist.itemlist.name in s.allitemlist.name
        all s:Storage|                                                  20
                all i,j:Item, c:ContainerItem|c in s.containerlist
                && j in c.itemlist && i in s.allitemlist
                        => i.getAmount[] = (sum c.getAmount[i.getName[]])
}
```

図 18: シグネチャStorage で表される倉庫の不変条件

図 18 にシグネチャStorage の不変条件による契約を示す．

最初の五つの式で倉庫内のコンテナに対し，品目は一種類以上もち，また，コンテナの数，品目の種類どちらもともに 0 であるか，0 でないかのどちらかであることを表す．次の式ではまた，品目の集合において，品名が一意であることを示している．また，コンテナの集合において，コンテナ ID が一意であることを示している．

また，最後の式に関しては，全品目の集合において，各品目ごとの総数は，各コンテナに含まれる同品目の総計であることを表している．

ここで用いた構文について軽く述べる．

$$all\ 変数名 : シグネチャ名 \mid 論理式 \tag{4}$$

これは，変数名で宣言された任意の変数に対し論理式が成り立つことを示す． $alli, j : Item|$ 〜 のように同時に複数の変数を扱うこともできる．また，

$$sum\ 式 \tag{5}$$

これは，全ての場合におけるその式の値の合計値を表す．

5.2.2 ContainerItem クラス

```
fact ContainerItem_Invariant{
        all c:ContainerItem|
                c.containerID>=0
        all c:ContainerItem|
                #c.itemlist>0
        all c:ContainerItem|
                c.carryingdate != none
        all  c:ContainerItem|
                all i,j: Item|
                        i in c.itemlist && j in c.itemlist−i                    10
                                => i.getName[]!=j.getName[] && #(i&c.itemlist) = 1
                                        && #(c.itemlist − i) = #c.itemlist−1
}
```

図 19: シグネチャContainerItem で表されるコンテナの不変条件

まず，コンテナ番号 ID は正であることを最初の文で表している．次の文では，コンテナは何らかの品目を最低一つは持っていなければならないことを表している．さらに，搬入日時は空ではない．

そして，Storage と同様に all 構文 4 を用いて，内蔵品リスト内に同じ品目を複数持たず，異なる二つの品目は，お互いを除いた内蔵品リストとの共通部分がないことを表している．

5.2.3 Item クラス

```
fact Item_Invariant{
        all i:Item|
                i.totalamount >= 0
                 && i.name != none
}
```

図 20: シグネチャItem で表されるコンテナの不変条件

totalamount で表される総個数が負ではないこと，また，名前が有効であることを表している．

5.2.4 Request クラス

```
fact Request_invariant{
        all r:Request|
        r.receiptiondate!=none
        all r:Request|
        r.itemname!=none
        all r:Request|
        r.amount > 0
        all r:Request|
        r.requeststate in StockState
        all r:Request|
        r.customer != none
}
```

図 21: シグネチャRequest で表されるコンテナの不変条件

注文としてもつそれぞれの要素が有効な値であることを示す．この fact 節中に出てくる StockState とは，注文が配送済みかそうでないかを表す状態を定義したシグネチャで，定数のようなものである．この場合，in 演算子を用いて，requeststate が StockState というシグネチャか，それを継承したシグネチャの集合に含まれることを表している．

5.2.5 Customer クラス

```
fact Customer_invariant{
        Customer.name != none
        Customer.address != none
        Customer.zipcode != none
        name != address
        address != zipcode
        zipcode != name
}
```

図 22: シグネチャCustomer で表されるコンテナの不変条件

顧客の個人情報がそれぞれ有効な値であることを表す．また，全て String というシグネチャの要素として宣言しているのだが，これらは同じものであってはならないので，全て違う値を持つことを表している．

5.3 各処理の記述

5.3.1 コンテナ受け付け処理

図 23 に倉庫でのコンテナ受付処理を Alloy でモデル化したコードを示す．まず，コンテナを受け付ける瞬間の倉庫の状態について考える．受け付けるコンテナが空ではなく，受け付けるコンテナの ID がコンテナの集合内のコンテナと重ならないことを示す述語を registContainerItem_pre として定義した．

次に，コンテナ受付を完了した時点での状態を考える．受け付け処理が終了した直後には，受け付けたコンテナがコンテナの集合に含まれていなければならない．このことを示す述語を registContainerItem_post として定義した．

ここで，上記の二つの述語が成り立つモデルそれぞれについて考える．s をコンテナ受け付け処理前の倉庫の状態であるとし，s' をコンテナ受付処理後の倉庫状態であるとするならば，倉庫である s と s' の差は，新しく登録されたコンテナである c がコンテナリストに含まれているかどうかということである．したがって，このメソッドの動作を示す述語は，registContainerItem_body のように記述できる．

また，この動作が正しく行われるかについては全ての倉庫とコンテナに関して上記の述語が成り立つことがいえれば良いから，図 23 での assert 節で示したようになる．

```
pred Storage.registContainerItem_pre[c:ContainerItem]{
        c != none
        all ci:ContainerItem|ci in this.containerlist
                => ci.getContainerID[] != c.getContainerID[]
}
pred Storage.registContainerItem_post[c:ContainerItem]{
        c in this.containerlist
}
pred registContainerItem_body[s,s':Storage,c:ContainerItem]{
                s.registContainerItem_pre[c] && s'.registContainerItem_post[c]
                        => s.containerlist = s'.containerlist - c
}
assert Storage_registContainerItem{
        all s:Storage, c:ContainerItem|some s':Storage|
                registContainerItem_body[s,s',c]
}
```

図 23: Alloy による Storage のコンテナ受け付け処理

5.3.2 不足在庫の確認処理

図 24 に倉庫での不足在庫の確認処理を Alloy でモデル化したコードを示す.

不足在庫の確認処理は，注文が送られてきたときに，注文品が在庫に含まれ，かつ，在庫が十分ある場合のみに真となる．このことを示す述語を checkStockSatisfied と定義する.

また，checkStockSatisfied の述語を通すためには，注文が空ではならないので，そのことを checkStockSatisfied_pre という述語で表した.

コンテナ受け付け処理とは違い，処理の前後で状態が変動しないため，この処理が行われたときの各引数にたいする出力を保障することを考える．入力である注文に示された注文品が在庫になければ checkStockSatisfied は真ではない．また，存在し，かつ在庫が足りているならば，真を返すが，そうでないならば偽である.

このような契約を表す述語は checkStockSatisfied_post として表した.

また，これによって，checkStockSatisfied_pre が成り立っているならば，そのどの場合に対しても出力がどうなるか定義されている checkStockSatisfied_post が成り立つことを表明できるので，assert 節にて表しておく.

```
pred Storage.checkStockSatisfied[r:Request]{
        r.getName[] in this.allitemlist.name
        some i:Item|i in this.allitemlist && i.getName[] = r.getName[]
                        && i.getAmount[] >= r.getAmount[]
}

pred Storage.checkStockSatisfied_pre[r:Request]{
        r != none
}

pred Storage.checkStockSatisfied_post[r:Request]{
        r.getName[] !in this.allitemlist.name => !this.checkStockSatisfied[r]
        r.getName[] in this.allitemlist.name
                && (all i:this.allitemlist|i in this.allitemlist
                && i.getName[] = r.getName[] && i.getAmount[] >= r.getAmount[])
                        => this.checkStockSatisfied[r]
                (all i:Item|i in this.allitemlist && i.getName[] = r.getName[]
                && i.getAmount[] < r.getAmount[])
                        => !this.checkStockSatisfied[r]
}

assert Storage_checkStockSatisfied{
        all s:Storage,r:Request|
                s.checkStockSatisfied_pre[r]=>s.checkStockSatisfied_post[r]
}
```

図 24: Alloy による Storage の不足在庫確認

5.3.3 出庫処理

まず，コンテナから注文で指定された分だけ出庫する処理について考える．この Alloy によるモデル記述は図 25 に示す．在庫は足りていても一つのコンテナからは出庫しきれないことがあるので，操作の呼び出し元に不足量を通知しなければならない．よって，注文であたえられた品目の数量とコンテナの残量の差を返すモデルを考え，shippingItem として，値を返す特殊な述語である fun を用いて表した．

また，この操作をするためには注文により与えられた品名が空ではなく，また，注文数量は正である必要があるので，それを shippingItem_pre という述語で表した．

また，与えられた品名と数量に対し，このモデルがどのような出力を返すかといったことを，処理直後の状態からみてどのような値を返しているかということと，その時の内部の値を保障するための述語をそれぞれ記述した．

品目が存在する場合のこのメソッドの出力を b として与えられているものとする．品目が存在しないときには与えられた数量をそのまま返す．このことを shippingItem_post_pre の述語で表した．

この出力が帰ったときのコンテナの中身について，出庫する対象の品目が内蔵品として存在し，かつ，shippingItem_post_pre の出力が内蔵品の数量を負にした値に等しければ，内蔵品の数量をその値にセットする Setter が呼び出され，正しく動作していたと言える．また，shippingItem_post_pre の値が正であったときは，指定された品目がこのコンテナから全部持ち出されているため，内蔵品の数量を 0 で，この状態に対して，内蔵品の残量を 0 にセットする Setter が正しく動作していると言える．また，このコンテナに要求品目が存在しなかった場合はそのまま要求量である a が不足量として返されることが言える．このことを述語 shippingItem_post_post にて表した．

これらの述語において，あるコンテナ c に対して事前条件が成り立つならば，出力がその通りであり，かつ，同じ引数を用いた場合の c' というコンテナに対して，b を c に対して出庫した場合の戻り値とすると，shippingItem_post_post を用いて，c' が c の事後状態とするならば，その出力が正しいということと，この時の c' のに含まれる要求品目の総量と shippingItem_post_pre の差が c に含まれる要求品目の数となる.，契約を最後の assert 節にて表明している．

```
fun ContainerItem.shippingItem[n:String,a:Int]:Int{
        this.itemlist.getName[]=n
                => a - this.getAmount[n] else a
}

pred ContainerItem.shippingItem_pre[n:String,a:Int]{
        n!=none
        a>0
}

fun ContainerItem.shippingItem_post_pre[n:String,a,b:Int]:Int{
        this.itemlist.getName[] = n
                => b else a
}

pred ContainerItem.shippingItem_post_post[n:String,a,b:Int]{
        all i:Item|Item in this.itemlist && i.getName = n &&
                this.shippingItem_post_pre[n,a,b] = 0-i.getAmount[]
                        => i.setAmount[0-this.shippingItem_post_pre[n,a,b]]
        all i:Item|Item in this.itemlist && i.getName[] = n &&
                this.shippingItem_post_pre[n,a,b] > 0 && i.getAmount[]=0
                        => i.setAmount[0]
        all i:Item|Item in this.itemlist && i.getName != n
                => this.shippingItem_post_pre[n,a,b]=a
}
assert ContainerItem_shippingItem{
        all c,c':ContainerItem,n:String,a:Int|
                c.shippingItem_pre[n,a] =>c.shippingItem[n,a] = a - c.getAmount[n]
                                || c.shippingItem[n,a] = a
                && let b=c.shippingItem[n,a] |c'.shippingItem_post_post[n,a,b]
                && c.getAmount[n] = c'.getAmount[n] - c'.shippingItem_post_pre[n,a,b]
}
```

図 25: Alloy による ContainerItem の搬出処理

5.4 Alloy による記述のまとめ

本研究において，Alloy では在庫管理問題の在庫のコンテナ受付や，要求品目をコンテナから出庫等，主要な動作の根幹に関わる部分について記述できた．

また，各オブジェクトにおける記述の量は表 6 のようになった．

しかし，受付業務や，出庫指示書の作成など，いくつかの部分においては記述できていない部分も存在する．

	LOC
Storage	134
ContainerItem	144
Request	117
Item	92
Customer	129
総計	621

表 6: Alloy による記述量

受け付け業務に関しては，受付ではなく倉庫に直接入力を与えることにより代替可能な処理ではあるので，この記述がなかったとしても在庫管理業務に関する処理の設計という点においては問題はないと言える．しかし，出庫指示書に関する契約が記述できてないことには，完全に設計ができたと言うことはできない．

この記述できていない部分が存在する原因として，Alloy にはループをサポートする構文，また，再帰をサポートする構文がないことが挙げられる．再帰に関しては代替手段が用意されているものの，繰り返し処理をすることにより判明する値が確実に入っている事を契約として記述することは現時点では難しい．この点の実現ができなかったため，本研究では出庫指示書の中身について保障することができなかった．また，同様の理由で，コンテナから品目を出庫する処理自体の設計はできても，その不足量をさらに要求量として，要求が満たされるまでそれぞれのコンテナから出庫させるといった契約を記述することができなかった．

6 Alloy Analyzer による検証

6.1 コンテナ受け付け処理

コンテナ受け付け処理の実体を表す registContainerItem_body のインスタンスを Alloy-Analyzer により図示させた．結果，図 26 に示されるように AlloyAnalyzer は，$p \Rightarrow q$ という論理式に対し,$\bar{p}$ というインスタンスを示した．また，assert 節で示した契約に対して，

```
No counterexample found. Storage_registContainerItem may be valid.
```

という出力を得ることができた.

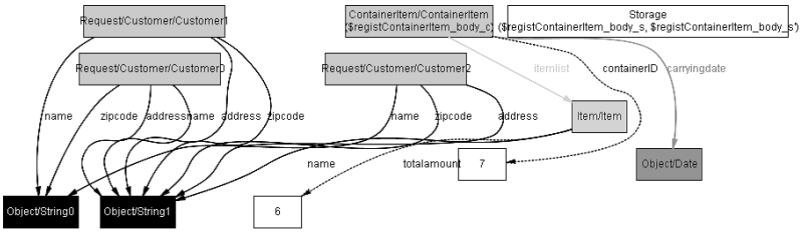

図 26: AlloyAnalyzer によるコンテナ受け付け処理検証

6.2 不足在庫の確認処理

処理の実体を示す，checkStockSatisfied に対してインスタンスを AlloyAnalyzer によって図示させた結果，図 27 のようになった．ここで示されたインスタンスは，名前 String2 の品目を 7 個要求し，それに対して在庫が 7 個存在する，真の値を返すインスタンスである．

また，assert 節で示した契約に対しては，

```
No counterexample found. Storage_checkStockSatisfied may be valid.
```

という出力を得ることができた.

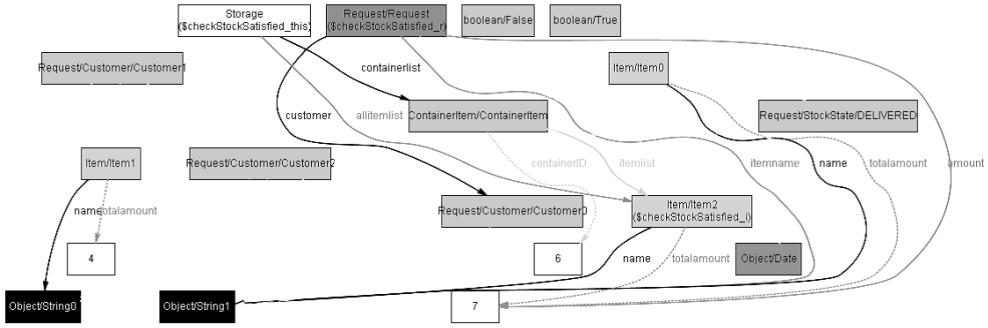

図 27: AlloyAnalyzer による不足在庫の確認処理検証

6.3 コンテナからの出庫処理

コンテナからの出庫処理に関しては，一つの述語から処理の実体を表すことができないので，assert 節で示される論理式に対して検証させた．結果，インスタンスは図 28 に表されるように出力され，出庫処理のための述語を組み合わせたインスタンスが存在することが示された．また，この assert 節に対する検証において，

```
No counterexample found. ContainerItem_shippingItem may be valid.
```

という出力を得ることができ，この処理の正当性が AlloyAnalyzer 的に証明された．

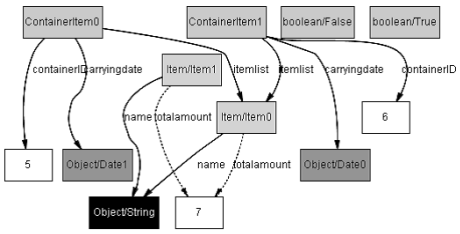

図 28: AlloyAnalyzer によるコンテナからの出庫処理検証

7　JML 記述と Alloy 記述の関連性

Alloy にて述語として示されるものをそのまま JML の事前条件，事後条件等で表すことは難しいが，Alloy にて JML 的な事前条件，事後条件を記述することは，以下の方法にて実現できる．

requires requires 節にて述べたい論理式を書いた述語に対し，それが成立するならば，処理開始直前の様子をモデル化した述語が成立する事を示す述語を用意することにより実現できる．

ensures メソッドの処理終了直後の様子をモデル化した述語を考える．処理直前の様子，処理直後の様子の述語を用いることにより\old に相当することを表すことができ，それを用いて ensures 節で述べたい論理式を書くことにより実現できる．

また，不変条件に関しては，JML においての invariant 節，Alloy における fact 節にて同様に表すことができる．以下，表 7 に同様の表現で表すことのできるものについてまとめた．

JML	Alloy
\forall	all
\exists	some
!(\exists〜;〜;〜)	no

表 7: JML と Alloy の構文の対応

8 あとがき

本稿では，在庫管理問題について，実装面に近い設計について JML で，高レベルでの設計を Alloy で行った．また，行った設計に対し，JML については ESC/Java を用いて，要求しように関連する各メソッドについて設計が正当であることを証明し，Alloy については AlloyAnalyzer を用いて，要求した動作を示すモデルのインスタンスの存在を示し，また，その契約について反例が存在しないことを示した．しかし，Alloy による設計については，一部のオブジェクトについてモデル化していないこともあり，完全とは言えず，その部分の設計は今後の課題である．

また，JML と Alloy による仕様記述において，お互いの類似点と相違点を整理し，関連性を提示することができた．

これらの成果を踏まえ，Alloy から JML 付きの Java のスケルトンコードへの変換手法の考案，また，それを用いた変換ツールの設計，実装，検証等が今後の課題として挙げられる．

謝辞

本研究において, 理解ある御指導を賜り，常に励まして頂きました 大阪大学大学院情報科学研究科楠本 真二 教授に心から感謝申し上げます.

本研究の全過程において, 終始適切な御指導および御助言を頂きました 岡野 浩三 助教授に心から感謝申し上げます.

本研究において, 常に適切な御指導および御助言を頂きました 山口 弘純 助手に深く感謝致します.

最後に, その他様々な御指導, 御助言を頂いた大阪大学 大学院情報科学研究科コンピュータサイエンス専攻 楠本研究室の皆様に深く感謝致します.

参考文献

[1] Gary T.Leavens, Yoonsik Cheon: “Design by Contract with JML,” September 28, 2006.

[2] Mandana Vaziri: “Finding Bugs in Software with a Constraint Solver,” Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis, pp.14-25, 2000.

[3] Sarfraz Khurshid, Darko Marinov: “TestEra A Novel Framework for Testing Java Program,” 16th IEEE Confe,”ence on Automated Software Engineering, pp22-31, November, 2001.

[4] Daniel Jackson: “Alloy: A Lightweight Object Modelling Notation,” ACM Transactions on Software Engineering and Methodology, Vol.11, Issue 2, pp.256-290, April, 2002.

[5] Bertrand Meyer: “Object-Oriented Software Construction, Second Edition,” Prentice Hall, 2000.

[6] Gary T. Leavens, Erik Poll, Curtis Clifton, Yoonsik Cheon, Clyde Ruby, David Cok, Peter Müller, and Joseph Kiniry: “JML Reference Manual (DRAFT),” 2007.

[7] Arun D. Raghavan and Gary T. Leavens: “Desugaring JML Method Specifications””, 2005.

[8] Gary T. Leavens, Albert L. Baker, and Clyde Ruby: “JML: A Notation for Detailed Design,” Behavioral Specifications of Businesses and Systems, pp175-188, 1999.

[9] Gary T. Leavens, Albert L. Baker, and Clyde Ruby: “Preliminary Design of JML: A Behavioral Interface Specification Language for Java.,” ACM SIGSOFT Software Engineering Notes, Vol.31, Issue 3, pp1-38, 2006.

[10] Patrice Chalin, Joseph R. Kiniry, Gary T. Leavens, and Erik Poll: “Beyond Assertions: Advanced Specification and Verification with JML and ESC/Java2,” Lecture Notes in Computer Science, Springer Verlag, Vol.4111, pp342-363, 2006.

[11] 大木優, 竹内彰一, 宮崎敏彦, 古川康一, 二村良彦:“Concurrent Prolog によるオンライン在庫管理システムの記述”, TM0082, 1984.

[12] 岡野浩三, 北道淳司, 東野輝夫, 谷口健一: “順序機械型プログラムの階層的設計法と在庫管理プログラムの開発例”, 電子情報通信学会論文誌, Vol.J76-DI, No.7, pp.354-363, 1993.

[13] 二村良彦, 雨宮真人, 山崎利治, 淵一博: “新しいプログラミング・パラダイムによる共通問題の設計”, 情報処理学会会誌 情報処理, Vol.25, No.5, 1985.

[14] MulSaw project: Suggested Extensions,
http://mulsaw.lcs.mit.edu/projects.html

[15] MIT Software Design Group, “The Alloy Analyzer,”
http://alloy.mit.edu/

[16] The Java Modeling Language,
http://www.cs.iastate.edu/ leavens/JML/

[17] Design by Contract - Wikipedia,
http://en.wikipedia.org/wiki/Design_by_contract

[18] Extended Static Checking for Java (from Internet archive)
http://web.archive.org/web/20051208055447/http://research.compaq.com/SRC/esc/

[19] Kind Software,
http://secure.ucd.ie/products/opensource/ESCJava2/

[20] 開発プロセスの基本を学ぶ,
http://itpro.nikkeibp.co.jp/article/lecture/20061130/255501/?ST=develop

付録

A　JML 付の Java による在庫管理プログラムソースコード

A-a　ReceiptionDesk.java

```
package StockManagement;

import java.util.*;

public class ReceiptionDesk {
        private /*@ spec_public non_null @*/ List RequestList;
        private /*@ spec_public non_null @*/ Storage storage;

        /*@ public invariant \typeof(RequestList) == \type(Request);
          @*/                                                          10


        /*@ public behavior
                requires r != null;
                assignable storage, RequestList;
                ensures \result == storage.checkStockSatisfied(r);
          @*/
        public boolean receiptRequest(Request r){
                if(storage.checkStockSatisfied(r)){
                        storage.deliveringOrder(r);                    20
                        return true;
                }
                if(RequestList.add(r)){
                        int tmpsize = RequestList.size();
                        Request tmplist[] = new Request[tmpsize];
                        Iterator i = RequestList.iterator();
                        for(int j=0;i.hasNext();j++){
                                tmplist[j] = (Request)i.next();
                        }
                        Arrays.sort(tmplist);                          30
                        RequestList.clear();
                        for(int j=0;j<tmpsize;j++){
                                RequestList.add(tmplist[j]);
                        }
                        return false;
                }
                else return false;
        }

        /*@ public behavior                                            40
                assignable RequestList,storage;
                ensures (\forall Request r; RequestList.contains(r)
```

```
                && (\sum ContainerItem ci;
                        \old(storage).deliveringOrder(r).contains(ci);
                        ((Item)ci.getContainedItem().get(0)).getAmount()
                )
                == r.getAmount();
                r.getRequestState() == StockState.DELIVERED
        );
        ensures (\forall Request r; RequestList.contains(r)
                && r.getRequestState() == StockState.SHORTAGE;
                \old(storage).checkStockSatisfied(r) == false);
        ensures (\forall Request r;RequestList.contains(r)
                && r.getRequestState() == StockState.DELIVERED
                && \old(r.getRequestState()) == StockState.SHORTAGE;
                \result.contains(\old(storage).deliveringOrder(r)));
  @*/
public List deliveringOrder()
{
        Iterator i = RequestList.iterator();
        List deliveredlist = new LinkedList();
        while(i.hasNext()){
                Request request = (Request)i.next();
                if(request.getRequestState()!=StockState.DELIVERED){
                        if(storage.checkStockSatisfied(request)){
                                deliveredlist.add(storage.deliveringOrder(request));
                                request.registAlreadyDelivered();
                        } else {
                                request.registShortageStock();
                                request.getCustomer().informShortageStock(request);
                        }
                }
        }
        return deliveredlist;
}


/*@ public behavior
        assignable \nothing;
        ensures \typeof(\result) == \type(Request);
  @*/
public /*@ pure @*/ List getRequestList()
{
        return RequestList;
}


/*@ public behavior
        ensures \result == storage;
```

```
        assignable \nothing;
  @*/
public /*@ pure @*/ Storage getStorage()
{
        return storage;
}


/*@ public behavior
        requires \typeof(storage.getItemList()) == \type(Item);
        assignable \nothing;
        ensures \result == storage.getItemList();
  @*/
public /*@ pure @*/ List getItemList()
{
        return storage.getItemList();
}


/*@ public behavior
        requires \typeof(storage.getContainerItemList())
                == \type(ContainerItem);
        assignable \nothing;
        ensures \result == storage.getContainerItemList();
  @*/
public /*@ pure @*/ List getContainerItemList()
{
        return storage.getContainerItemList();
}


/*@ public behavior
        requires c != null;
        assignable storage;
        ensures (\exists ContainerItem ci;
                storage.getContainerItemList().contains(ci);
                ci.getContainerID()==c.getContainerID());
  @*/
public void receiptNewContainerItem(ContainerItem c)
{
        ContainerItem citem =
                new ContainerItem(c.getContainerID(),
                        c.getContainedItem(),
                        c.getReceiptionDate()
                );
        storage.registContainerItem(c);
        deliveringOrder();
```

```
    }

    /*@ public behavior
            assignable RequestList, storage;                                    140
      @*/
    public ReceiptionDesk()
    {
            RequestList = new LinkedList();
            storage = new Storage();
    }
}
```

A-b Storage.java

```
package StockManagement;
import java.io.*;
import java.util.*;
public class Storage{
    private /*@ spec_public non_null @*/
            LinkedList containerlist = new LinkedList();
    private /*@ spec_public non_null @*/
            LinkedList allitemlist = new LinkedList();

    /*@ public invariant \typeof(containerlist) == \type(ContainerItem);          10
            public invariant \typeof(allitemlist) == \type(Item);
            public invariant (\forall Item i,j;allitemlist.contains(i)
                    && allitemlist.contains(j)
                    && i!=j;!i.getName().equals(j.getName()));
            public invariant (\forall ContainerItem ci, cj;
                    containerlist.contains(ci) && containerlist.contains(cj) && ci!=cj;
                    ci.getContainerID()!=cj.getContainerID());
      @*/

    /*@ public behavior                                                         20
            requires c != null;
            requires (\forall ContainerItem i; containerlist.contains(i);
                    i.getContainerID()!=c.getContainerID());
            requires (\forall Item i,j; c.getContainedItem().contains(i)
                    && c.getContainedItem().contains(j);
                    !i.getName().equals(j.getName()));
            assignable containerlist, allitemlist;
            ensures containerlist.contains(c);
            ensures (\forall ContainerItem i;\old(containerlist).contains(i);
                    i.getContainerID()!=c.getContainerID());                    30
            ensures (\forall Item i,j; c.getContainedItem().contains(i)
                    && allitemlist.contains(j) && \old(j.getName().equals(i.getName()));
                            j.getAmount() == \old(j.getAmount()+i.getAmount())
            );
```

```
        ensures (\forall Item i,j; c.getContainedItem().contains(i)
                && allitemlist.contains(j) && !\old(allitemlist).contains(j);
                j.getAmount() == i.getAmount()
        );
        ensures !\old(allitemlist.isEmpty())
                ==> !allitemlist.equals(c.getContainedItem());

  @*/
public void registContainerItem(ContainerItem c)
{
        containerlist.add(c);
        updateAllItemList(c);
        return;
}

/*@ public behavior
        requires c != null;
        requires (\forall Item i,j; c.getContainedItem().contains(i)
                && c.getContainedItem().contains(j);
                !i.getName().equals(j.getName()));
        assignable allitemlist;
        ensures (\forall Item i,j; c.getContainedItem().contains(i)
                && allitemlist.contains(j) && \old(j.getName().equals(i.getName()));
                        j.getAmount() == \old(j.getAmount()+i.getAmount())
        );
        ensures (\forall Item i,j; c.getContainedItem().contains(i)
                && allitemlist.contains(j)
                && !\old(allitemlist).contains(j) && j.getName().equals(i.getName());
                j.getAmount() == i.getAmount()
        );
        ensures !\old(allitemlist.isEmpty())
                ==> !allitemlist.equals(c.getContainedItem());
  @*/
private void updateAllItemList(ContainerItem c)
{
        Iterator i = c.getContainedItem().iterator();
        /*@
                maintaining i != null && i.hasNext();
                maintaining (\sum Item item;allitemlist.contains(item);item.getAmount())
                        == \old((\sum Item item;allitemlist.contains(item);
                                item.getAmount()))
                        +(\sum Item item;c.getContainedItem().contains(item);
                                item.getAmount());
          @*/
        while(i.hasNext()){
                Item itemi = (Item) i.next();
                Iterator j = allitemlist.iterator();
```

```
                /*@
                        maintaining j != null &&j.hasNext();
                        maintaining itemi != null && itemi.getAmount()>0;
                        maintaining (\exists Item it;allitemlist.contains(it)
                                &&it.getName().equals(itemi.getName());
                                it.getAmount()
                                        == \old(it.getAmount())+itemi.getAmount());
                  @*/
                while(j.hasNext()){                                              90
                        Item itemj = (Item) j.next();
                        if((itemj.getName()).equals(itemi.getName())){
                                itemj.setAmount(itemj.getAmount()
                                        + itemi.getAmount());
                                break;
                        }
                }
                /*@
                        assert (\exists Item it;allitemlist.contains(it)
                                &&it.getName().equals(itemi.getName());          100
                                it.getAmount()
                                        == \old(it.getAmount())+itemi.getAmount());
                  @*/
                if(!j.hasNext()){
                        Item tmpitem = new Item(itemi.getName(),itemi.getAmount());
                        allitemlist.add(tmpitem);
                }
        }
        /*@
                assert (\sum Item item;allitemlist.contains(item);               110
                        item.getAmount())
                        == \old((\sum Item item;
                                allitemlist.contains(item);item.getAmount()))
                                + (\sum Item item;
                                        c.getContainedItem().contains(item);
                                        item.getAmount()
                                );
          @*/
        return;
}                                                                                120

/*@ public behavior
        ensures containerlist.containsAll(\result);
        ensures (\forall ContainerItem ci; \result.contains(ci);
                (\forall Item i; ci.getContainedItem().contains(i); i.getAmount()==0)
        );
        assignable \nothing;
  @*/
```

```
public LinkedList informEmptyContainer()
{                                                                          130
        LinkedList tmp = new LinkedList();
        Iterator i = containerlist.iterator();
        while(i.hasNext()){
                boolean b=true;
                ContainerItem citem = (ContainerItem)i.next();
                Iterator j = citem.getContainedItem().iterator();
                while(j.hasNext()){
                        Item item = (Item)j.next();
                        b &= (item.getAmount()==0);
                }                                                          140
                if(b) tmp.add(citem);
        }
        return tmp;
}

/*@ public behavior
        requires r != null;
        assignable \nothing;
        ensures \result == (\exists Item i;allitemlist.contains(i);
                r.getName().equals(i.getName())                            150
                && r.getAmount() <=i.getAmount());
  @*/
public boolean checkStockSatisfied(Request r)
{
        Iterator i = allitemlist.iterator();
        while(i.hasNext()){
                Item item = (Item)i.next();
                if(r.getName().equals(item.getName())
                        && r.getAmount()<=item.getAmount()){
                        return true;                                       160
                }
        }
        return false;
}

/*@ public behavior
        requires r != null;
        requires checkStockSatisfied(r);
        assignable allitemlist, containerlist;
        ensures (\sum ContainerItem i; \result.contains(i);                170
                ((Item)i.getContainedItem().get(0)).getAmount()) == r.getAmount();
        ensures (\forall ContainerItem i;\result.contains(i);
                ((Item)i.getContainedItem().get(0)).getName().equals(r.getName()));
  @*/
public List deliveringOrder(Request r)
```

```
{
        int shortageamount = r.getAmount();
        int n = r.getAmount();
        List DeliveredList = new LinkedList();
        Iterator i = allitemlist.iterator();                                    180

        while(i.hasNext()){
                Item item = (Item)i.next();
                if(item.getName().equals(r.getName())){
                        item.setAmount(item.getAmount() - r.getAmount());
                        Iterator j = containerlist.iterator();
                        while(j.hasNext()){
                                Item delivereditem;
                                ContainerItem containeritem
                                        = (ContainerItem)j.next();              190
                                n = containeritem.shippingItem(r.getName()
                                        ,r.getAmount());
                                if(n>0){
                                        delivereditem =
                                        new Item(r.getName(),
                                                containeritem.getAmount(r.getName()));
                                }else{
                                        delivereditem = new Item(r.getName(),-n);
                                }
                                List cil = new LinkedList();                    200
                                cil.add(delivereditem);
                                ContainerItem c =
                                        new ContainerItem(
                                                containeritem.getContainerID(),
                                                ci,
                                                new Date(r.getReceiptionDate())
                                        );
                                DeliveredList.add(c);
                                if(n<=0) break;
                        }                                                       210
                        break;
                }
        }
        return DeliveredList;
}

/*@ public behavior
        ensures \result == allitemlist;
        assignable \nothing;
  @*/                                                                           220
public /*@ pure @*/ List getItemList()
{
```

```
            return allitemlist;
        }


        /*@ public behavior
            ensures \result == containerlist;
            assignable \nothing;
          @*/                                                          230
        public /*@ pure @*/ List getContainerItemList()
        {
            return containerlist;
        }


        /*@ public behavior
            assignable \nothing;
            ensures containerlist.isEmpty();
            ensures allitemlist.isEmpty();                             240
          @*/
        public Storage()
        {
            //containerlist = new LinkedList();
            //allitemlist = new LinkedList();
        }
}
```

A-c ContainerItem.java

```
package StockManagement;
import java.util.*;
import java.util.*;
public class ContainerItem {
        private /*@ spec_public non_null @*/ Date carryingDate;
        private int containerID;
        private /*@ spec_public non_null @*/ LinkedList itemlist;
        /*@ public invariant containerID >= 0;
            public invariant \typeof(itemlist) == \type(Item);
            public invariant (\forall Item i,j;                        10
                    itemlist.contains(i) && itemlist.contains(j)
                    && i != j;i.getName() !=j.getName());
          @*/

        /*@ public behavior
            requires name != null && !name.equals("");
            requires num > 0;
            ensures (\exists Item i; itemlist.contains(i);
                    i.getName().equals(name)&&\result == \old(num-i.getAmount()));
            assignable itemlist;                                       20
```

```
  @*/
public int shippingItem(String name, int num)
{
        Iterator i = itemlist.iterator();
        int tmp = num;
        while(i.hasNext()){
                Item item = (Item) i.next();
                System.out.println(item.getName() + " , " + name);
                if((item.getName()).equals(name)){
                        tmp -= item.getAmount();                                    30
                        if(item.getAmount() > num) {
                                item.setAmount(item.getAmount()-num);
                        } else {
                                item.setAmount(0);
                        }
                        return tmp;
                }
        }
        return tmp;
}                                                                                   40

/*@ public behavior
        ensures \result == containerID;
        assignable \nothing;
  @*/
public /*@ pure @*/ int getContainerID()
{
        return containerID;
}
                                                                                    50

/*@ public behavior
        requires name != null && !name.equals("");
        assignable \nothing;
        ensures (\exists Item i;itemlist.contains(i);
                i.getAmount() == \result) || \result == 0;
  @*/
public /*@ pure @*/ int getAmount(String name)
{
        Iterator i=itemlist.iterator();                                             60
        while(i.hasNext()){
                Item item = (Item)i.next();
                if(item.getName().equals(name)){
                        return item.getAmount();
                }
        }
        return 0;
```

```
}

/*@ public behavior
        ensures \result == carryingDate;
        assignable \nothing;
  @*/
public /*@ pure @*/ Date getReceiptionDate()
{
        return carryingDate;
}

/*@ public behavior
        ensures \result == itemlist;
        assignable \nothing;
  @*/
public /*@ pure @*/ List getContainedItem(){
        return itemlist;
}


/*@ public behavior
        requires ID >= 0;
        requires il != null
                &&      \typeof(il) == \type(Item);
        requires (\forall Item i,j;il.contains(i)
                && il.contains(j) && il.indexOf(i)!=il.indexOf(j);
                !i.getName().equals(j.getName())
        );
        assignable containerID, itemlist, carryingDate;
        ensures containerID == ID;
        ensures itemlist.equals(il);
        ensures carryingDate.equals(d);
  @*/
public ContainerItem(int ID, List il, Date d)
{
        //long date = d.getTime();
        containerID=ID;
        itemlist = new LinkedList(il);
        carryingDate = d;
}

/*@ public behavior
        requires ID >= 0;
        requires il != null
```

```
                &&      \typeof(itemlist) == \type(Item);
        requires (\forall Item i,j;il.contains(i)
                && il.contains(j) && il.indexOf(i)!=il.indexOf(j);
                !i.getName().equals(j.getName())
        );
        assignable containerID, itemlist, carryingDate;
        ensures containerID == ID;
        ensures itemlist.equals(il);
        ensures carryingDate == new Date();
  @*/
public ContainerItem(int ID, List il)
{
        containerID=ID;
        itemlist = new LinkedList(il);
        carryingDate = new Date();
}


/*@
        requires ID >= 0;
        assignable containerID, itemlist, carryingDate;
        ensures containerID == ID;
        ensures itemlist.equals(new LinkedList());
        ensures carryingDate != null;
  @*/
public ContainerItem(int ID)
{
        containerID=ID;
        itemlist = new LinkedList();
        carryingDate = new Date();
}

/*@
        also
        public normal_behavior
        assignable \nothing;
        ensures \result.matches("ContainerID."
                + containerID + "CarryingDate : " + carryingDate + "\n{1}");
        ensures (\forall Item i; itemlist.contains(i);
                \result.matches("^\t" + i + "\n{1}"));
  @*/
public String toString()
{
        Iterator i = itemlist.iterator();
        StringBuffer tmp = new StringBuffer("ContainerID."
                + containerID + "CarryingDate : " + carryingDate + "\n");
        while(i.hasNext()){
```

```
                Item item = (Item)i.next();
                tmp.append("\t" + item + "\n");
            }
            return tmp.toString();
        }

}
```

A-d Request.java

```
package StockManagement;
import java.util.*;
public class Request implements Comparable{
        private /*@ spec_public non_null @*/
                Date receiptionDate;
        private
                Date deliveringDate;
        private /*@ spec_public non_null @*/
                String itemName;
        private /*@ spec_public @*/                                    10
                int amount;
        private
                byte requestState;
        private /*@ spec_public non_null @*/
                Customer customer;

        /*@ public invariant amount>0; @*/


        /*@ public behavior                                            20

                requires requestState != StockState.DELIVERED;
                assignable requestState;
                ensures requestState == StockState.SHORTAGE;
          @*/
        public void registShortageStock()
        {
                requestState = StockState.SHORTAGE;
        }
                                                                       30

        /*@ public behavior

                requires requestState != StockState.SHORTAGE;
                assignable requestState;
                ensures requestState == StockState.DELIVERED;
          @*/
        public void registAlreadyDelivered()
```

```
{
        requestState = StockState.DELIVERED;
}

/*@ public behavior
        ensures \result == customer;
  @*/
public /*@ pure @*/ Customer getCustomer()
{
        return customer;
}

/*@ public behavior
        requires num>0;
        assignable amount;
        ensures amount == num;
  @*/
public void setAmount(int num)
{
        amount = num;
}

/*@ public behavior
        ensures \result == amount;
        assignable \nothing;
  @*/
public /*@ pure @*/ int getAmount()
{
        return amount;
}

/*@ public behavior
        ensures \result == requestState;
        assignable \nothing;
  @*/
public /*@ pure @*/ byte getRequestState()
{
        return requestState;
}

/*@ public behavior
        ensures \result == receiptionDate.getTime();
```

```
        assignable \nothing;
  @*/
public /*@ pure @*/ long getReceiptionDate()
{
        return receiptionDate.getTime();                                    90
}


/*@ also
        public normal_behavior
        requires o instanceof Request;
        assignable \nothing;
        ensures \result ==
                (int)(receiptionDate.getTime() - ((Request)o).getReceiptionDate());
  @*/                                                                       100
public /*@ pure @*/ int compareTo(Object o)
{
        if(o instanceof Request)
        {
                return (int)(receiptionDate.getTime()
                        - ((Request)o).getReceiptionDate()
                );
        } else {
                return 0;
        }                                                                   110
}


/*@ public behavior
        ensures \result.equals(itemName);
        assignable \nothing;
@*/
public /*@ pure @*/ String getName()
{
        return itemName;                                                    120
}


/*@ public behavior
        requires rd!=null;
        requires iname != null && iname.equals("");
        requires c != null;
        requires am>0;
        assignable receiptionDate,itemName,amount,requestState,customer;
        ensures amount==am;                                                 130
        ensures itemName.equals(iname);
        ensures requestState == rqst;
```

```
        ensures receiptionDate.equals(rd);
        ensures customer.equals(c);
  @*/
public Request(String iname, int am, Customer c, Date rd, byte rqst)
{
        receiptionDate = rd;
        itemName = new String(iname);
        amount = am;
        requestState = rqst;
        customer = c;
}


/*@ public behavior
        requires rd!=null;
        requires iname != null && iname.equals("");
        requires c != null;
        requires am>0;
        assignable receiptionDate,itemName,amount,requestState,customer;
        ensures amount==am;
        ensures itemName.equals(iname);
        ensures requestState == StockState.DELIVERED;
        ensures receiptionDate.equals(rd);
        ensures customer.equals(c);
  @*/
public Request(String iname, int am, Customer c, Date rd)
{
        receiptionDate = rd;
        itemName = new String(iname);
        amount = am;
        requestState = StockState.DELIVERED;
        customer = c;
}


/*@ public behavior
        requires iname != null && iname.equals("");
        requires c != null;
        requires am>0;
        assignable receiptionDate,itemName,amount,requestState,customer;
        ensures amount==am;
        ensures itemName.equals(iname);
        ensures requestState == StockState.DELIVERED;
        ensures receiptionDate != null;
        ensures customer == c;
  @*/
public Request(String iname, int am, Customer c)
```

```
        {                                                                    180
                receiptionDate = new Date();
                itemName = new String(iname);
                amount = am;
                requestState = StockState.DELIVERED;
                customer = c;
        }
}
```

A-e Item.java

```
package StockManagement;
import java.util.*;
public class Item{
        private /*@ spec_public non_null @*/ String name;
        private /*@ spec_public @*/ int totalamount;

        /*@ public invariant totalamount >= 0
                && name != null && !name.equals("");
          @*/
                                                                             10


        /*@ public behavior
                ensures \result.equals(name);
                assignable \nothing;
          @*/
        public /*@ pure @*/ String getName()
        {
                return new String(name);
        }                                                                    20


        /*@ public behavior
                ensures \result == totalamount;
                assignable \nothing;
          @*/
        public /*@ pure @*/ int getAmount()
        {
                return totalamount;
        }                                                                    30


        /*@ public behavior
                requires num > 0;
                assignable totalamount;
                ensures totalamount == num;
          @*/
```

```
        public void setAmount(int num)
        {
                totalamount = num;                                              40
        }


        /*@ also
                public normal_behavior
                assignable \nothing;
                ensures \result.equals("ItemName : "
                        + name + " , amount : " + totalamount);
          @*/
        public String toString()                                                50
        {
                return "ItemName : " +name + " , amount : " + totalamount;
        }


        /*@ public behavior
                requires n != null && !n.equals("") && a > 0;
                assignable name,totalamount;
                ensures name.equals(n);
                ensures totalamount == a;                                       60
          @*/
        public Item(String n, int a)
        {
                name = new String(n);
                totalamount = a;
        }
}
```

A-f Customer.java

```
package StockManagement;
import java.util.*;
public class Customer {
        private /*@ spec_public non_null @*/
                String name;
        private /*@ spec_public non_null @*/
                String address;
        private /*@ spec_public non_null @*/
                String zipcode;
                                                                                10
        /*@ public invariant !name.equals("");
                public invariant !address.equals("");
                public invariant !zipcode.equals("");
          @*/
        /*@ public behavior
```

```
        requires r != null;
        assignable \nothing;
        ensures \result == true;
  @*/
public /*@ pure @*/ boolean informShortageStock(Request r)          {
        return true;
}


/*@ public behavior
        ensures \result.equals(name);
        assignable \nothing;
  @*/
public /*@ pure @*/ String getName()
{
        return name;
}


/*@ public behavior
        ensures \result.equals(address);
        assignable \nothing;
  @*/
public /*@ pure @*/ String getAddress()
{
        return address;
}


/*@ public behavior
        ensures \result.equals(zipcode);
        assignable \nothing;
  @*/
public /*@ pure @*/ String getZipCode()
{
        return zipcode;
}


/*@ public normal_behavior
        requires c != null;
        ensures \result == name.equals(c.getName())
                && address.equals(c.getAddress())
                && zipcode.equals(c.getZipCode());
        assignable \nothing;
  @*/
public boolean equals(Customer c){
```

```
        return name.equals(c.getName()) && address.equals(c.getAddress())
                && zipcode.equals(c.getZipCode());
}


/*@ public behavior
        requires nm != null && !nm.equals("");
        requires ad != null && !ad.equals("");                                  70
        requires zc != null && !zc.equals("");
        assignable name,address,zipcode;
        ensures name.equals(nm)
                && address.equals(ad) && zipcode.equals(zc);
  @*/
public Customer(String nm, String ad, String zc)
{
        name = new String(nm);
        address = new String(ad);
        zipcode = new String(zc);                                               80
}


/*@ public behavior
        assignable name,address,zipcode;
        ensures name!= null && address!= null && zipcode!= null;
  @*/
public Customer()
{
        name = new String("John Doe");                                          90
        address = new String("Nothing");
        zipcode = new String("Nothing");
}


/*@ public behavior
        requires nm != null && !nm.equals("");
        assignable name;
        ensures name.equals(nm);
  @*/                                                                           100
public void setName(String nm)
{
        name = new String(nm);
}


/*@ public behavior
        requires ad != null && !ad.equals("");
        assignable address;
```

```
        ensures address.equals(ad);
    @*/
    public void setAddress(String ad)
    {
        address = new String(ad);
    }


    /*@ public behavior
        requires zc != null && !zc.equals("");
        assignable zipcode;
        ensures zipcode.equals(zc);
    @*/
    public void setZipCode(String zc)
    {
        zipcode = new String(zc);
    }
}
```

B Alloy による在庫管理プログラムソースコード

B-a Storage.als

```
module AlloyStockManagement/Storage
open util/boolean as boolean
open ElementTypes as Object
open AlloyStockManagement/ContainerItem as ContainerItem
open AlloyStockManagement/Item as Item
open AlloyStockManagement/Request as Request

sig Storage{
    allitemlist:set Item,
    containerlist:set ContainerItem
}

fact Storage_Invariant{
    all s:Storage,c:ContainerItem|
        c in s.containerlist => #c.itemlist > 0
    all s:Storage|
        #s.containerlist > 0 => #s.allitemlist>0
    all s:Storage|
        #s.allitemlist > 0 => #s.containerlist>0
    all s:Storage|
        #s.containerlist = 0 => #s.allitemlist=0
    all s:Storage|
        #s.allitemlist=0 => #s.containerlist=0
    all s:Storage,i,j:Item|
        i in s.allitemlist && j in (s.allitemlist−i)
```

```
                    => i.getName[] != j.getName[] && #(s.allitemlist & i) = 1
        all s:Storage,i,j:ContainerItem|
                i in s.containerlist && j in (s.containerlist-i)
                    => i.getContainerID[]!=j.getContainerID[]
        all s:Storage|                                                          30
                s.containerlist.itemlist.name in s.allitemlist.name
        all s:Storage|
                all i,j:Item, c:ContainerItem|c in s.containerlist
                && j in c.itemlist && i in s.allitemlist
                    => i.getAmount[] = (sum c.getAmount[i.getName[]])
}

pred Storage.registContainerItem_pre[c:ContainerItem]{
        c != none
        all ci:ContainerItem|ci in this.containerlist                           40
                => ci.getContainerID[] != c.getContainerID[]
}

pred Storage.registContainerItem_post[c:ContainerItem]{
        c in this.containerlist
}

pred registContainerItem_body[s,s':Storage,c:ContainerItem]{
                s.registContainerItem_pre[c]
                    && s'.registContainerItem_post[c]                           50
                    => s.containerlist = s'.containerlist - c
}
assert Storage_registContainerItem{
        all s:Storage, c:ContainerItem|some s':Storage|
                registContainerItem_body[s,s',c]
}

pred Storage.updateAllItemList_pre[c:ContainerItem]{
        c != none
        c in this.containerlist                                                 60
}

pred Storage.updateAllItemList_post[c:ContainerItem]{
        c.itemlist.name in this.allitemlist.name
        c in this.containerlist
}

pred updateAllItemList_body[s,s':Storage,c:ContainerItem]{
        s.updateAllItemList_pre[c] && s'.updateAllItemList_post[c]
                => c.itemlist.name in s'.allitemlist.name                       70
}
```

```
assert Storage_updateAllItemList{
        all s:Storage, c:ContainerItem|some s':Storage|
                updateAllItemList_body[s,s',c]
}

pred Storage.checkStockSatisfied[r:Request]{
        r.getName[] in this.allitemlist.name
        some i:Item|i in this.allitemlist && i.getName[] = r.getName[]
                        && i.getAmount[]>=r.getAmount[]
}

pred Storage.checkStockSatisfied_pre[r:Request]{
        r != none
}

pred Storage.checkStockSatisfied_post[r:Request]{
        r.getName[] !in this.allitemlist.name
                => !this.checkStockSatisfied[r]
        r.getName[] in this.allitemlist.name
                && (all i:this.allitemlist|i in this.allitemlist
                        && i.getName[] =r.getName[]
                        && i.getAmount[] >= r.getAmount[])
                =>this.checkStockSatisfied[r]
                (all i:Item|i in this.allitemlist
                        && i.getName[] = r.getName[]
                        && i.getAmount[] < r.getAmount[])
                =>!this.checkStockSatisfied[r]
}

assert Storage_checkStockSatisfied{
        all s:Storage,r:Request|
                s.checkStockSatisfied_pre[r]=>s.checkStockSatisfied_post[r]
}

/*fun Storage.deliveringOrder[r:Request]:Int{
        r.getName[] in this.containerlist.name
                =>
}*/
pred Storage.deliveringOrder_pre[r:Request]{
        this.checkStockSatisfied_pre[r] => this.checkStockSatisfied_post[r]
}
pred Storage.deliveringOrder_post[r:Request]{
        some i:Item|i in this.allitemlist && i.getAmount[]>=0
}
pred deliveringOrderModel[s,s':Storage,r:Request]{
                s.deliveringOrder_pre[r]=>s'.deliveringOrder_post[r]
                s != s'
```

```
                s' in s                                                    120
                all i,j:Item|i in s.allitemlist
                        && j in s'.allitemlist && i.getName[]=j.getName[]
                        && j.getName[] =r.getName[]
                        => i.getAmount[] = j.getAmount[]+r.getAmount[]
}
assert Storage_deliveringOrder{
        all s:Storage,r:Request|some s':Storage|
                deliveringOrderModel[s,s',r]
                        => s != s' && s' in s
                        && all i,j:Item|i in s.allitemlist                 130
                        && j in s'.allitemlist && i.getName[]=j.getName[]
                        && j.getName[] =r.getName[]
                                => i.getAmount[] = j.getAmount[]+r.getAmount[]
}

pred show[]{}


check Storage_registContainerItem
check Storage_updateAllItemList                                            140
check Storage_checkStockSatisfied
check Storage_deliveringOrder
run deliveringOrderModel
run {some s:Storage, r:Request|s.checkStockSatisfied[r]}
run registContainerItem_pre
run updateAllItemList_body
run checkStockSatisfied
run show
```

B-b ContainerItem.als

```
module AlloyStockManagement/ContainerItem
open util/boolean as boolean
open ElementTypes as Object
open AlloyStockManagement/Item as Item

sig ContainerItem{
        carryingdate:Date,
        containerID:Int,
        itemlist:some Item
}                                                                          10

fact ContainerItem_Invariant{
        all c:ContainerItem|
                c.containerID>=0
        all c:ContainerItem|
                #c.itemlist>0
```

```
        all c:ContainerItem|
                c.carryingdate != none
        all   c:ContainerItem|
                all i,j: Item|
                        i in c.itemlist && j in c.itemlist−i
                                => i.getName[]!=j.getName[] &&#(i&c.itemlist) = 1
                                        && #(c.itemlist − i) = #c.itemlist−1
}

fun ContainerItem.shippingItem[n:String,a:Int]:Int{
        this.itemlist.getName[]=n
                => a − this.getAmount[n] else a
}

pred ContainerItem.shippingItem_pre[n:String,a:Int]{
        n!=none
        a>0
}

fun ContainerItem.shippingItem_post_pre[n:String,a,b:Int]:Int{
        this.itemlist.getName[] = n
                => b else a
}

pred ContainerItem.shippingItem_post_post[n:String,a,b:Int]{
        all i:Item|Item in this.itemlist && i.getName = n &&
                this.shippingItem_post_pre[n,a,b] = 0−i.getAmount[]
                        => i.setAmount[0−this.shippingItem_post_pre[n,a,b]]
        all i:Item|Item in this.itemlist && i.getName[] = n &&
                this.shippingItem_post_pre[n,a,b] > 0 && i.getAmount[]=0
                        => i.setAmount[0]
        all i:Item|Item in this.itemlist && i.getName != n
                => this.shippingItem_post_pre[n,a,b]=a
        //for visualyze
        //some i:Item|Item in this.itemlist => i.getName = n && a>0
}
assert ContainerItem_shippingItem{
        all c,c':ContainerItem,n:String,a:Int|
                c.shippingItem_pre[n,a] =>c.shippingItem[n,a] = a - c.getAmount[n]
                                || c.shippingItem[n,a] = a
        //all c:ContainerItem,n:String,a:Int|
        &&      let b=c.shippingItem[n,a] |c'.shippingItem_post_post[n,a,b]
        && c.getAmount[n] = c'.getAmount[n] - c'.shippingItem_post_pre[n,a,b]
}

fun ContainerItem.getContainerID[]:Int{
        this.containerID
```

```
}

assert ContainerItem_getContainerID{
        all c:ContainerItem|
                c.containerID=c.getContainerID[]
}

fun ContainerItem.getAmount[n:String]:Int{
        this.itemlist.getName=n=>this.itemlist.getAmount[] else 0
}

pred ContainerItem.getAmount_post[n:String]{
        some i:Item|i in this.itemlist=>i.getName[]=n
                        =>i.getAmount[] = this.getAmount[n]
        else this.getAmount[n] = 0
}

assert ContainerItem_getAmount{
        all c:ContainerItem, n:String| c.getAmount_post[n]
}

fun ContainerItem.getContainedItem[]:Item{
        this.itemlist
}

pred ContainerItem.getContainedItem_post[]{
        this.getContainedItem[] = this.itemlist
}

assert ContainerItem_getContainedItem{
        all c:ContainerItem|c.getContainedItem_post[]
}

fun ContainerItem.getReceiptionDate[]:Date{
        this.carryingdate
}

pred ContainerItem.getReceiptionDate_post[]{
        this.getReceiptionDate[]=this.carryingdate
}

assert ContainerItem_getReceiptionDate{
        all c:ContainerItem|c.getReceiptionDate_post[]
}

pred ContainerItem.Constructor[id:Int,il:Item,date:Date]{
        this.containerID=id
```

```
        this.itemlist= il
        this.carryingdate=date
}

pred ContainerItem.Constructor_pre[id:Int,il:Item,date:Date]{
        id>=0
        #il > 0
        date != none
        all i,j: Item|
                i in il && j in il-i                                    120
                        => i.getName[]!=j.getName[] &&#(i&il) = 1
                                && #(il - i) = #il-1
}

assert ContainerItem_Constructor{
        all c:ContainerItem,id:Int,il:Item,date:Date|
                c.Constructor[id,il,date] =>c.Constructor_pre[id,il,date]
}

pred show[]{}                                                           130

check ContainerItem_shippingItem
check ContainerItem_getContainerID
check ContainerItem_getAmount
check ContainerItem_getContainedItem
check ContainerItem_getReceiptionDate
check ContainerItem_Constructor

run shippingItem
run shippingItem_pre                                                    140
run shippingItem_post_pre
run shippingItem_post_post
run {   all c,c':ContainerItem,n:String,a:Int|
                c.shippingItem_pre[n,a] =>c.shippingItem[n,a] = a - c.getAmount[n]
                        || c.shippingItem[n,a] = a
        //all c:ContainerItem,n:String,a:Int|
        &&      let b=c.shippingItem[n,a] |c'.shippingItem_post_post[n,a,b]
        && c.getAmount[n] = c'.getAmount[n] - c'.shippingItem_post_pre[n,a,b]
}
run Constructor                                                         150
run show
```

B-c Request.als

```
module AlloyStockManagement/Request
open util/boolean as boolean
open ElementTypes as Object
open AlloyStockManagement/Customer as Customer
```

```
open AlloyStockManagement/StockState as StockState

sig Request{
        receiptiondate : one Date,
        deliveringdate : lone Date,
        itemname : one String,                                          10
        amount : one Int,
        requeststate : StockState,
        customer : one Customer
}

fact Request_invariant{
        all r:Request|
        r.receiptiondate!=none
        all r:Request|
        r.itemname!=none                                                20
        all r:Request|
        r.amount > 0
        all r:Request|
        r.requeststate in StockState
        all r:Request|
        r.customer != none
}

pred Request.Constructor[n:String,a:Int,c:Customer,d:Date,s:StockState]{
        this.itemname=n                                                 30
        this.amount=a
        this.customer=c
        this.receiptiondate=d
        this.requeststate=s
}


assert Request_Constructor{
        all r:Request, n:String,a:Int,c:Customer,d:Date,s:StockState|
                n=none || a<1 || c=none || d=none || s=none             40
                        => !r.Constructor[n,a,c,d,s]
        all r:Request, n:String,a:Int,c:Customer,d:Date,s:StockState|
                r.Constructor[n,a,c,d,s]=>r.itemname=n &&
                        r.amount=a &&
                        r.customer=c &&
                        r.receiptiondate=d &&
                        r.requeststate=s
}

fun Request.getName[]:String{                                           50
        this.itemname
```

```
}

assert Request_getName{
        all r:Request|
                r.itemname = r.getName[]
}

fun Request.getAmount[]:Int{
        this.amount
}

assert Request_getAmount{
        all r:Request|
                r.amount = r.getAmount[]
}

fun Request.getRequestState[]:StockState{
        this.requeststate
}

assert Request_getRequestState{
        all r:Request|
                r.requeststate = r.getRequestState[]
}

fun Request.getReceiptionDate[]:Date{
        this.receiptiondate
}

assert Request_getReceiptionDate{
        all r:Request|
                r.receiptiondate=r.getReceiptionDate[]
}

pred Request.registShortageStock[]{
        this.requeststate=SHORTAGE
}

assert Request_registShortageStock{
        // post condition
        all r:Request|
                r.registShortageStock[]
                        =>r.getRequestState = SHORTAGE
}

pred Request.registAlreadyDelivered[]{
        this.requeststate=DELIVERED
```

```
}

assert Request_registAlreadyDelivered{
        // post condition
        all r:Request|
                r.registAlreadyDelivered[]
                        =>r.getRequestState = DELIVERED
}

pred show[]{}

check Request_Constructor
check Request_getName
check Request_getAmount
check Request_getRequestState
check Request_getReceiptionDate
check Request_registShortageStock
check Request_registAlreadyDelivered
run show
```

B-d Item.als

```
module AlloyStockManagement/Item
open util/boolean as boolean
open ElementTypes as Object

sig Item{
        name:one String,
        totalamount:one Int
}

fact Item_Invariant{
        all i:Item|
                i.totalamount >= 0
                 && i.name != none
}

pred Item.Constructor[n:String, a:Int]{
        this.name=n
        this.totalamount=a
}

pred Item.Constructor_pre[n:String,a:Int]{
        n != none
        a > 0
}
```

```
assert Item_Constructor{
        all i:Item,n:String,a:Int|
                (i.Constructor_pre[n,a]=>i.Constructor[n,a])
                        => i.Constructor[n,a]=>i.getName[]=n && i.getAmount=a
}

fun Item.checkStockSatisfying_body[n:Int]:Bool{
        this.totalamount>=n => True else False
}

pred Item.checkStockSatisfying_pre[n:Int]{
        n>0
}

pred Item.checkStockSatisfying_post[n:Int]{
        this.totalamount>=n => this.checkStockSatisfying_body[n]=True
        this.totalamount<n => this.checkStockSatisfying_body[n]=False
}

assert Item_checkStockSatisfying{
        all i:Item,n:Int|
                i.checkStockSatisfying_pre[n]=>i.checkStockSatisfying_post[n]
}

fun Item.getName[]:String{
        this.name
}

assert Item_getName{
        all i:Item|
                i.name = i.getName[]
}

fun Item.getAmount[]:Int{
        this.totalamount
}

assert Item_getAmount{
        all i:Item|
                i.totalamount = i.getAmount[]
}

pred Item.setAmount[a:Int]{
        this.totalamount=a
}

pred Item.setAmount_pre[a:Int]{
```

```
        a >= 0
}

assert Item_setAmount{
        all i:Item,a:Int|
                a>0=> i.setAmount_pre[a]
        all i:Item, a:Int|
                i.setAmount[a]=>i.totalamount=a
}

pred show[]{}

check Item_Constructor
check Item_checkStockSatisfying
check Item_getName
check Item_getAmount
check Item_setAmount
run getAmount
run setAmount
run Constructor
```

B-e Customer.als

```
module AlloyStorageManagement/Customer

open util/boolean as boolean
open ElementTypes as Object
sig Customer{
        name:one String,
        address:one String,
        zipcode:one String
}

fact Customer_invariant{
        Customer.name != none
        Customer.address != none
        Customer.zipcode != none
        name != address
        address != zipcode
        zipcode != name
}

pred Customer.Constructor[n,a,z:String]{
        this.name=n
        this.address=a
        this.zipcode=z
}
```

```
assert Customer_Constructor{
        all c:Customer,n,a,z:String|
                n=none || a = none || z = none
                        => !c.Constructor[n,a,z]
        all c:Customer,n,a,z:String|                                    30
                Customer.Constructor[n,a,z]
                        => c.name=n &&c.address=a &&c.zipcode=z
}

pred Customer.equals[c:Customer]{
        this.name=c.getName &&
        this.address=c.getAddress &&
        this.zipcode=c.getZipCode
}
                                                                        40
/*fun Customer.equalsBody[c:Customer] : Bool{
        this.name=c.getName &&
        this.address=c.getAddress &&
        this.zipcode=c.getZipCode
                => True else False
}*/

assert Customer_equals{
        all c1,c2:Customer|c1!=none && c2 !=none
                && c1.name = c2.name                                    50
                && c1.address=c2.address
                && c1.zipcode = c2.zipcode
                => c1.equals[c2]
}

pred Customer.setName[n:String]{
        this.name = n
}

assert  Customer_setName{                                               60
        all c:Customer, n:String |
                n= none => !c.setName[n]
        all c:Customer, n:String |
                c.setName[n] => c.name = n
}

fun Customer.getName[]:String{
        this.name
}
                                                                        70
assert Customer_getName{
        all c:Customer| c.name !=none
```

```
        all c:Customer|c.getName[]=c.name
}

pred Customer.setAddress[a:String]{
        this.address = a
}

assert  Customer_setAddress{                                                    80
        all c:Customer, a:String |
                a= none => !c.setAddress[a]
        all c:Customer, a:String |
                c.setAddress[a] => c.address = a
}

fun Customer.getAddress[]:String{
        this.address
}
                                                                                90
assert Customer_getAddress{
        all c:Customer| c.address !=none
        all c:Customer|c.getAddress[]=c.address
}

pred Customer.setZipCode[z:String]{
        this.zipcode= z
}

assert Customer_setZipCode{                                                     100
        all c:Customer, z:String |
                z= none => !c.setZipCode[z]
        all c:Customer, z:String|
                c.setZipCode[z] => c.zipcode=z
}

fun Customer.getZipCode[]:String{
        this.zipcode
}
                                                                                110
assert Customer_getZipCode{
        all c:Customer| c.zipcode !=none
        all c:Customer| c.getZipCode = c.zipcode
}

pred show[]{}

check Customer_getName
check Customer_setName
```

check Customer_getAddress
check Customer_setAddress

check Customer_getZipCode
check Customer_setZipCode

check Customer_Constructor
check Customer_equals
run show
