U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

Formal Modeling of a Fashion Show in VDM++

Mestrado Integrado em Engenharia Informática e Computação

Métodos Formais em Engenharia de Software

Elementos do Grupo:

Catarina Ramos - up201406219

Diogo Campos - up201403468

Contents

1. Informal system description and list of requirements

1.1 Informal system description

The model allows to manage different shows that can take place in different spaces with several stages. A show has a start date, an end date and a theme. It's possible to manage shows with the same theme but only if the dates don't overlap. The space can be reserve by several shows as long as the dates don't overlap. It's possible to know the participating or attending designers of a show or event.

A show can have multiple events between the show's start and end date. An event can take place in a stage of the space that the show has reserved as long as there is no other event in that stage between the event start and end time. It's possible to add and remove workers of a show and notify the workers/attendees about that event. An event can be a runway, in which there is a sequence of models, or a presentation where there is a discussion or a speech by one or more person, which can be a worker at the event or a guest at the show.

A person can attendee several shows by buying a ticket of a category, one ticket per show. A person can only have a guest ticket if the show as invited him. A person can design clothes that are presented in a runway or make speeches in presentations as long he is a guest at the show. It's possible to reserve entrance in a event that belongs to a show in which the person has a ticket to, to manage the jobs he has at the events and have access to the notifications of a job or event.

1.2 List of requirements

1.2.1 Person

A person can play different roles in simultaneous regarding the fashion shows, therefore, each role has different requirements. The roles are:

- Designer: When a person has designed clothes;
- Worker: When a person has at least one job at a show's event;
- Attendee: When a person has a ticket to a show:
    - Vip
    - Guest
    - Normal

Id	Priority	Description
R1	Mandatory	As a person, I can buy a ticket of a show if there are still tickets left for the ticket type that I want to buy.
R2	Low	As a person, I can set my basic information.

1.2.1.1 Designer

Id	Priority	Description
R3	Medium	As a designer, I can be a participant designer at an event if I have a job at a runway and at least one of the models is using my cloths.
R4	Mandatory	As a designer, I can design new clothes and added them to my portfolium.

1.2.1.1 Worker

Id	Priority	Description
R5	Mandatory	As a worker, I have access to all the notifications targeted to the workers for the events I work in.
R6	Medium	As a worker, I can be dismissed from an event I work on.
R7	Medium	As a worker, I can be contracted to an event only if I am available.

1.2.1.1 Attendee

Id	Priority	Description
R8	Mandatory	As a guest/normal attendee, I can reserve my attendance at an event only if there are still seats left.
R9	Mandatory	As a vip attendee, I can reserve my attendance at an event without the need of making a reservation.
R10	Mandatory	As a guest attendee, I don’t have to reserve my attendance at the presentation that I am going to be a speaker.
R11	Mandatory	As an attendee,I can cancel an attendance reservation.
R12	Mandatory	As an attendee,I can cancel my speech at a presentation.
R13	Mandatory	As an attendee,I can consult the events that I have the attendance reservation per show or the next to attend after a certain date.
R14	Mandatory	As an attendee,I can have access to all the designers that are participating/attending an event/show.

1.2.1 Manager

Id	Priority	Description
R15	Mandatory	As a manager, I can manage new shows.
R16	Medium	As a manager, I can remove a show that was being managed.
R17	Mandatory	As a manager, I can sell tickets to the show and control the limit of tickets to sold per type.
R18	Mandatory	As a manager, I can add/cancel events to/from a show.

R19	Mandatory	As a manager, I can contract/dismiss workers for a certain event.
R20	Low	As a manager, I can set the basic information about the shows and events that I manage.
R21	Medium	As a manager, I can invite guests to be (or not) a speaker at a presentation.
R22	Medium	As a manager, I can set the sequence of a runway such has the clothes and models (workers) used.
R23	Medium	As a manager, I can add speakers to a presentation.
R24	Mandatory	As a manager, I can control the reservations of an event.
R25	Medium	As a manager, I can add notifications to an event targeting the workers or attendees.

1.2.3 Space Owner

Id	**Priority**	**Description**
R26	Medium	As a space owner, I have access to all the shows that are going to take place in the space that I rent.
R27	Mandatory	As a space owner, I can add/remove stages to/from the space and set their information and capacity.
R28	Mandatory	As a space owner, I can rent or cancel a rent of the space to/from a show.

These requirements are directly translated onto use cases as shown next.

2. Visual UML model [1]

2.1 Use case model [2]

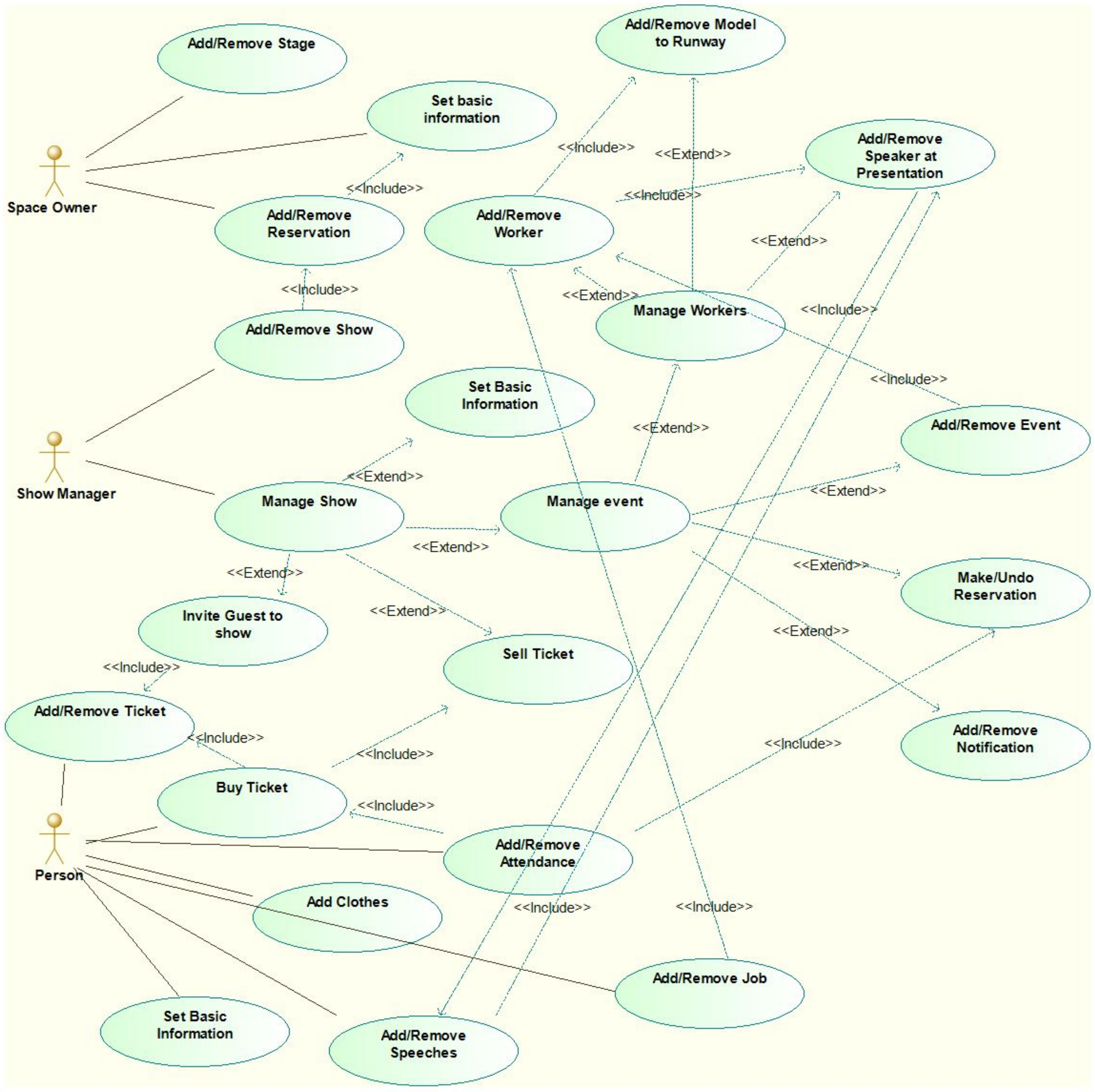

Image 1 - Use case model for the Fashion Show system

<u>Note:</u> The “adds” and “removes” are joined so that the use case becomes simpler to analyse.

[1] **Tip**: Diagrams can be modeled with Modelio 3.2 to take advantage of the integration with Overture.
[2] **Tip**: Coarse grained use cases, representing user goals, were decomposed into fine grained use cases, corresponding to simpler user interactions (to be modeled as operations).

The major use case scenarios (to be used later as test scenarios) are described next. There were operations implemented to perform these case scenarios.

Scenario	**Buy a Ticket**
Description	Normal scenario for a person to buy a ticket of a certain type for a certain show.
Pre-conditions	1. The show must have tickets left for the type of ticket. 2. The person must not have a ticket (of any kind) for that show. 3. Ticket type must not be of the type Guest
Post-conditions	1. The show sold a ticket of a certain type. 2. The person must have a ticket to that show in its name. 3. The show are in the “attendance” of the person
Steps	1. Create a ticket in the name of the person targeting the desired show 2. Add the ticket to the show’s tickets sold by type. 3. Add the ticket to the person tickets.
Exceptions	(none)

Scenario	**Attend an Event**
Description	Normal scenario for a person to reserve its entrance to an event
Pre-conditions	1. The person must have a ticket for the show that is going to host the desired event 2. There are still seats left to make an attendance reservation on the event. 3. The person doesn’t have an attendance reservation already for the event
Post-conditions	1.The event is in the “attendances” of the person 2. The number of seats reserved was incremented
Steps	1. Add the event to the attendances (by show)
Exceptions	(none)

Scenario	**Manage a new show**
Description	Normal scenario for a manager to manage a new show
Pre-conditions	1. The show can’t overlap with other shows that are going to take place in the same space. 2. The manager can’t have shows with the same theme and overlapping dates
Post-conditions	1. The new show was added to the managed shows.
Steps	1. Create the show 2. Add the show to the managed shows
Exceptions	(none)

Scenario	**Create an event and add it to the show**
Description	Normal scenario for a manager to add a new event to a show
Pre-conditions	1. The new event can’t overlap other events that are going to take place in the same stage at overlapping times. 2. The stage of the event must belongs to the space where the show is going to take place 3. The event must start and end between the show’s dates

Post-conditions	1. The event was created with the correct information 2. The event was added to the show
Steps	1. Create an Event 2. Added it to the event
Exceptions	(none)

Scenario	**Invite a guest to speak at a presentation**
Description	Normal scenario for a manager to invite a guest to speak at a presentation
Pre-conditions	1. The guest has to be free during the time of the presentation. 2. The guest can't have a ticket for this show already.
Post-conditions	1. A guest ticket was sold to the desired person.
Steps	1. Create a presentation. 2. Add it to the show. 3. Invite guest to speak at the presentation.
Exceptions	(none)

2.2 Class model

The image below represents the class model of the information system. The methods of the classes are omitted so that the class diagram is easier to analyse. The complete information about each class can be viewed in the appendix.

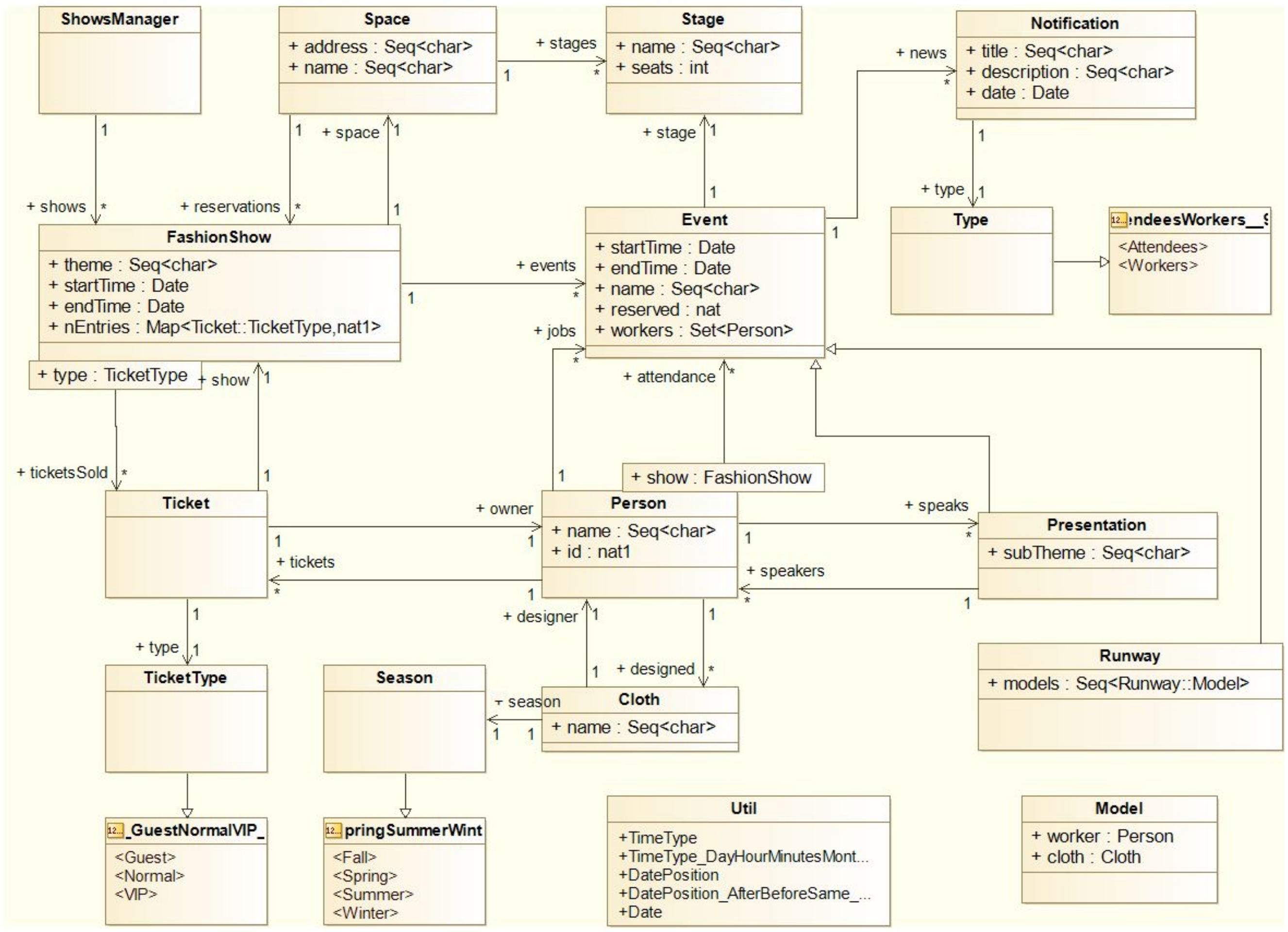

Image 2 - Simple class diagram (without the methods)

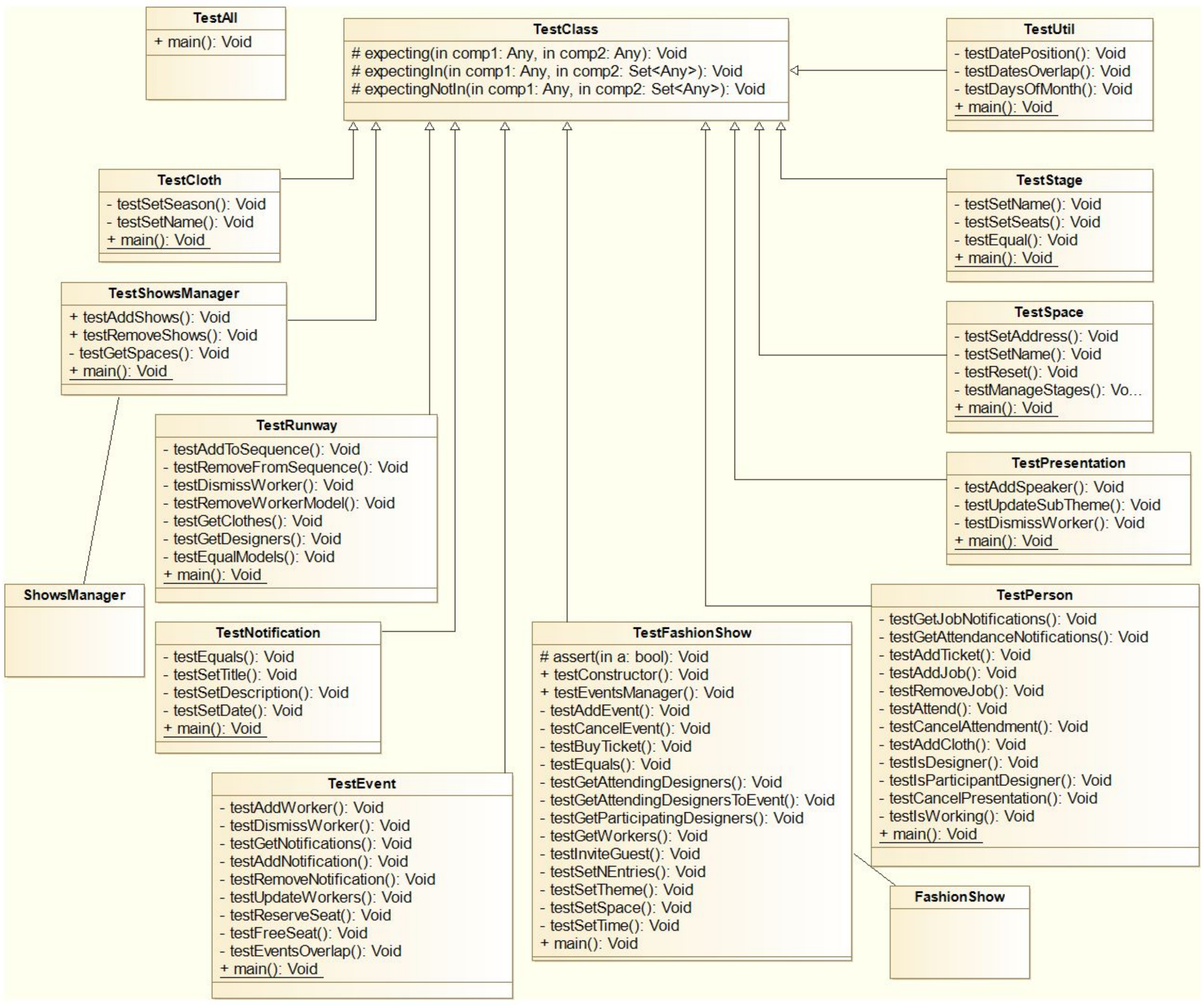
TestAll
+ main(): Void
TestClass
# expecting(in comp1: Any, in comp2: Any): Void
# expectingIn(in comp1: Any, in comp2: Set<Any>): Void
# expectingNotIn(in comp1: Any, in comp2: Set<Any>): Void
TestUtil
- testDatePosition(): Void
- testDatesOverlap(): Void
- testDaysOfMonth(): Void
+ main(): Void
TestCloth
- testSetSeason(): Void
- testSetName(): Void
+ main(): Void
TestStage
- testSetName(): Void
- testSetSeats(): Void
- testEqual(): Void
+ main(): Void
TestShowsManager
+ testAddShows(): Void
+ testRemoveShows(): Void
- testGetSpaces(): Void
+ main(): Void
TestSpace
- testSetAddress(): Void
- testSetName(): Void
- testReset(): Void
- testManageStages(): Vo...
+ main(): Void
TestRunway
- testAddToSequence(): Void
- testRemoveFromSequence(): Void
- testDismissWorker(): Void
- testRemoveWorkerModel(): Void
- testGetClothes(): Void
- testGetDesigners(): Void
- testEqualModels(): Void
+ main(): Void
TestPresentation
- testAddSpeaker(): Void
- testUpdateSubTheme(): Void
- testDismissWorker(): Void
+ main(): Void
ShowsManager
TestNotification
- testEquals(): Void
- testSetTitle(): Void
- testSetDescription(): Void
- testSetDate(): Void
+ main(): Void
TestFashionShow
# assert(in a: bool): Void
+ testConstructor(): Void
+ testEventsManager(): Void
- testAddEvent(): Void
- testCancelEvent(): Void
- testBuyTicket(): Void
- testEquals(): Void
- testGetAttendingDesigners(): Void
- testGetAttendingDesignersToEvent(): Void
- testGetParticipatingDesigners(): Void
- testGetWorkers(): Void
- testInviteGuest(): Void
- testSetNEntries(): Void
- testSetTheme(): Void
- testSetSpace(): Void
- testSetTime(): Void
+ main(): Void
TestPerson
- testGetJobNotifications(): Void
- testGetAttendanceNotifications(): Void
- testAddTicket(): Void
- testAddJob(): Void
- testRemoveJob(): Void
- testAttend(): Void
- testCancelAttendment(): Void
- testAddCloth(): Void
- testIsDesigner(): Void
- testIsParticipantDesigner(): Void
- testCancelPresentation(): Void
- testIsWorking(): Void
+ main(): Void
TestEvent
- testAddWorker(): Void
- testDismissWorker(): Void
- testGetNotifications(): Void
- testAddNotification(): Void
- testRemoveNotification(): Void
- testUpdateWorkers(): Void
- testReserveSeat(): Void
- testFreeSeat(): Void
- testEventsOverlap(): Void
+ main(): Void
FashionShow

Image 3 - Simple Test class diagram

Class	Description
ShowsManager	Class that allows to manage multiple shows.
FashionShow	Class that represents a show and has methods to manage the events.
Space	Class that represents a physical space where a show can take place.
Stage	Class that represents a division in a space where an event can take place.
Event	Class that represents an event and has methods to manage the event activity.
Presentation	Subclass of Event. Represents a specific type of event where there are speakers.
Runway	Subclass of Event. Main type of event of a fashion show.
Cloth	Class that represents a cloth.
Season	Class that represents a value of the enum _FallSprintSummerWinter_.

Ticket	Class that represents the access of a person who to a fashion show.
TicketType	Class that represents a value of the enum _GuestNormalVIP_.
Person	Class that represents a person and has methods to interacts with a fashion show. A person can interact with a fashion show by being an attendee, a designer, a worker or a speaker.
Model	Class that associates a worker (model) to a cloth. In VDM++ it is represented as a type.
Notification	Class that represents the news about an event. It can be targeting the workers or the attendees.
Type (Notification)	Class that represents a value of the enum _AttendeesWorkers_.
Util	Class with utilitary methods and types.

Enum	**Description**
GuestNormalVIP	Enum that represent the type of entrance in a fashion show. In VDM++ it was represented as a type inside the class Ticket.
FallSpringSummerWinter	Enum season of the year. Used by the class Cloth to established the season that the cloth was targeted. In VDM++ it was represented as a type inside the class Cloth.
AttendeesWorkers	Enum that represents the target of a notification. In VDM++ was it was represented as a type inside the class Notification.

Note: In the generation of the UML, the types defined inside a class were converted to classes. Therefore, the enumerations were converted to a class and an enumeration in the uml, where the class was a subclass of the enum.

3. Formal VDM++ model

Model, with corresponding coverage, is in the annexed PDF file at the end of the Appendix.

4. Model validation

The model validation includes the acceptance test for the use cases defined in the sections 1.2 and 2.1 and also unitary tests for each class. The test classes are those who start by “Test”. These tests are in the same annexed PDF file as the Formal Model (at the end of the Appendix).

5. Model verification

5.1 Example of domain verification

One of the proof obligations generated by Overture is:

No.	PO Name	Type
37	FashionShow`sellTicket(Person,Ticket`TicketType)	legal map application

The code under analysis (with the relevant map application underlined) is:

```
 --SellTicket
 public sellTicket: Person * Ticket`TicketType ==> Ticket
 sellTicket(attendee, type) == (
         dcl t :Ticket := new Ticket(type,attendee, self);
         ticketsSold := ticketsSold ++ {type |-> (ticketsSold(type) union {t})};
         return t
 )
 pre nEntries(type) > card ticketsSold(type) and                --PRE: Has entries for ticket type
                 type <> <Guest>                                --PRE: Can't buy guest tickets
 post (card ticketsSold~(type) + 1 = card ticketsSold(type));   --POS: Ticket was sold
```

The proof obligation view:

```
(forall attendee:Person, type:Ticket`TicketType & (((nEntries(type) > (card ticketsSold(type))) and
(type <> <Guest>)) => (type in set (dom ticketsSold))))
```

However, the invariant (in the class Fashion Show):

```
inv <Guest> in set dom ticketsSold and <VIP> in set dom ticketsSold and <Normal> in set dom
ticketsSold;
```

And the Ticket`TicketType values:

```
public static TicketType = <Normal> | <VIP> | <Guest>;
```

Guarantees that all the Ticket`TicketType values are keys in the map ticketsSold. Therefore, since the argument “type” is a Ticket`TicketType, this invariante assures that the map is accessed only inside its domain.

5.2 Example of invariant verification

One of the proof obligations generated by Overture is:

No.	PO Name	Type
78	Person`addTicket(Ticket)	state invariant holds

The code under analysis (with the relevant state change underlined) is:

```
--Add Ticket
public addTicket: Ticket ==> ()
addTicket(t) == (
	atomic(
		tickets := tickets union {t};
		attendance := attendance ++ {t.show |-> {}}
	)
)
pre t.owner = self and not exists t2 in set tickets & t.show = t2.show
--PRE: Person is owner of ticket
post (t in set tickets and t.show in set dom attendance);
--POS: New Ticket in tickets and show in attendance schedule (per show)
```

The relevant invariants under analysis are:

1. inv forall t in set tickets & t.show in set dom attendance;	--INV all tickets shows are in attendance
2. inv forall s in set dom attendance & exists t in set tickets & t.show = s;	--INV: Attendances only in events belonging to a show with a ticket
3. inv forall t in set tickets & t.owner.id = id;	--INV: All tickets addressed to the person

The 3. invariant becomes irrelevant inside the body of the operation since the precondition already restricts the operation only if the ticket is addressed to the person.

The atomic operation assures that the invariants are check at the end of all attributions. Therefore, the 1. and 2. invariant have to be fulfilled by the assignments inside the atomic operation.

Formally the post condition after the block is:

```
t in set tickets and t.show in set dom attendance
```

The condition above as to implies the 1. and the 2. invariant:

```
t in set tickets and t.show in set dom attendance => (forall t in set tickets & t.show in set dom
attendance) and (forall s in set dom attendance & exists t in set tickets & t.show = s)
```

This implication is true.

6. Code Generation

The java code was generated with success, however, when opening the created project, there were some errors when compiling. Below are the solutions to solve these problems:

1. In the function *cancelAttendment* in the class Person:
   a. Change last if condition:

      ```
      if (Utils.is_(event, Presentation.class)) {
      cancelPresentation(event);
      }
      ```

   b. By:

      ```
      if (Utils.is_(event, Presentation.class)) {
      cancelPresentation((Presentation)event);
      }
      ```

2. In the function *getParticipatingDesigners* in the class FashionShow:
   a. Change the if condition:

      ```
      if (Utils.is_(e, Runway.class)) {
       designers = SetUtil.union(Utils.copy(designers), e.getDesigners());
      }
      ```

   b. By:

      ```
      if (Utils.is_(e, Runway.class)) {
       designers = SetUtil.union(Utils.copy(designers),((Runway) e).getDesigners());
      }
      ```

To run the tests follow the steps bellow

1. Change the class TestAll to:

```
package generated;
import java.util.*;
import org.overture.codegen.runtime.*;

@SuppressWarnings("all")

public class TestAll {
 public static void main(String[] args) {
         System.out.println("Testing class Cloth ...");
         new TestCloth().main();
         System.out.println("Testing class Event ...");
```

```
            new TestEvent().main();
            System.out.println("Testing class FashionShow ...");
            new TestFashionShow().main();
            System.out.println("Testing class Notification ...");
            new TestNotification().main();
            System.out.println("Testing class Person ...");
            new TestPerson().main();
            System.out.println("Testing class Presentation ...");
            new TestPresentation().main();
            System.out.println("Testing class Runway ...");
            new TestRunway().main();
            System.out.println("Testing class ShowsManager ...");
            new TestShowsManager().main();
            System.out.println("Testing class Space ...");
            new TestSpace().main();
            System.out.println("Testing class Stage ...");
            new TestStage().main();
            System.out.println("Testing class Util ...");
            new TestUtil().main();
            System.out.println("\nTests completed!");

 }

 public TestAll() {}

 public String toString() {
   return "TestAll{}";
 }

}
```

2. In the class Utils change the functions *expecting* to:

```
public static void expecting(final Object p1, final Object p2) {
 if(p1 != p2) throw new NullPointerException();
 return;
}

public static void expecting(final Boolean b1, final Boolean b2) {
if(b1 != b2) throw new NullPointerException();
 return;
}
```

3. Open the run configurations and set to run the class TestAll as the entry point;
4. Run TestAll;

When running TestAll, all tests that were created in the VDM (and that were converted into java classes) are being executed. However, the pre and post conditions are not generated when generating the java code and, as a consequence, the code isn't going to throw an error when it behaves wrongly.

One solution to overcome partially this problem is by changing the functions as in the point 2. (above). The functions are used by the tests so,by doing this, when running the tests, every time the the *expecting* function is called, its assured that if it doesn't behaves correctly, then it will throw an exception. This solution is only partially because when interacting with the system it can have behave wrongly without throwing any errors. A way to have a solution that overcomes the problem in its totality would be to implement the pre and post conditions in the java and throw exceptions if they are not accepted.

7. Conclusions

For this project, we fully covered all the requirements specified. However, since the model is flexible enough to be more complex, there could have been some improvements and other requirements that could have been implemented with more time.

Some improvements could have been testing the model in a more exhaustive way and add more complexity to the model. For the latter part, the events could have more types, for exemple: designers meetings. There could also be implemented a store for the clothes presented in the runways.

Although the work was divided by the two elements of the group fairly, there was a slight difference in terms of contribution to the final result. As such, the contribution for each element was:

- Catarina Ramos: 60%
- Diogo Campos: 40%

The time invested for the implementation was approximately 50 hours.

8. References

1. Validated Designs for Object-oriented Systems, J. Fitzgerald, P.G. Larsen, P. Mukherjee, N. Plat, M. Verhoef, Springer, 2005
2. Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014
3. Tool web site, http://overturetool.org

Appendix

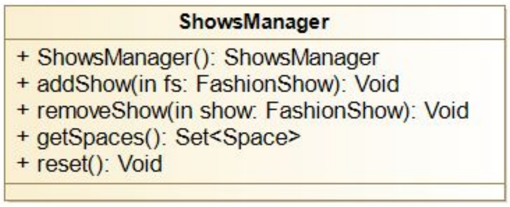
ShowsManager
+ ShowsManager(): ShowsManager
+ addShow(in fs: FashionShow): Void
+ removeShow(in show: FashionShow): Void
+ getSpaces(): Set<Space>
+ reset(): Void

Image 4 - Class Shows Manager

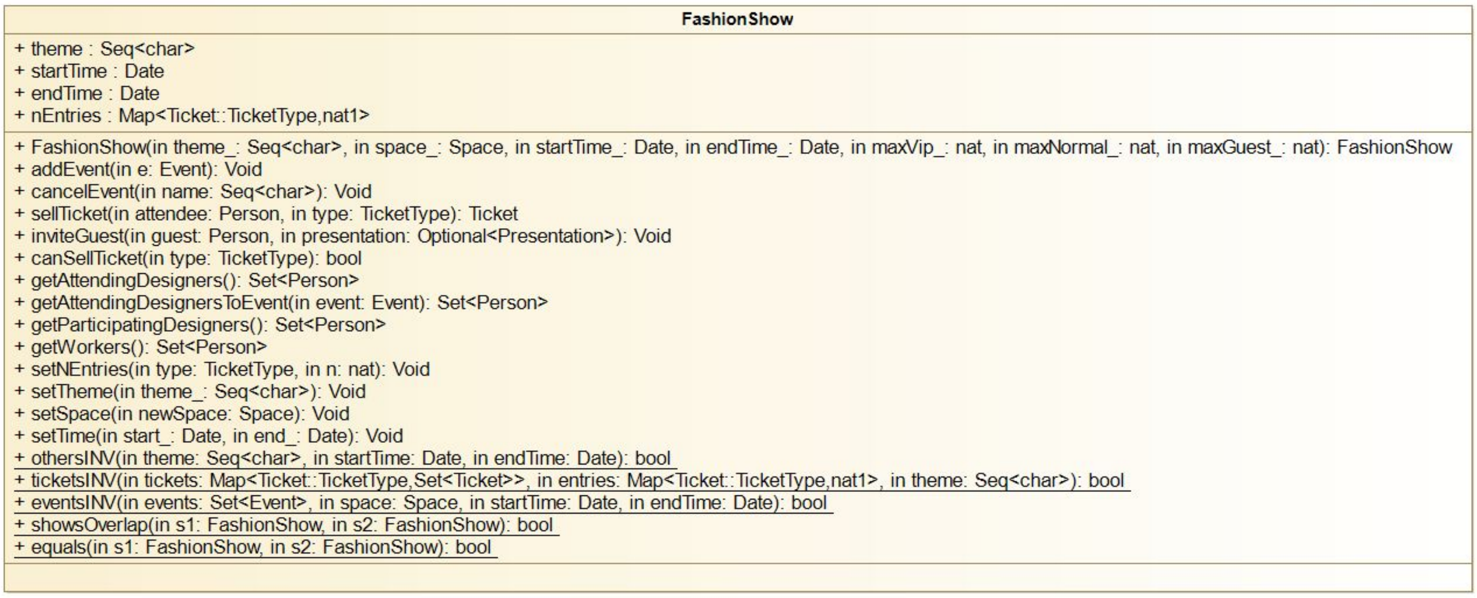
FashionShow
+ theme : Seq<char>
+ startTime : Date
+ endTime : Date
+ nEntries : Map<Ticket::TicketType,nat1>
+ FashionShow(in theme_: Seq<char>, in space_: Space, in startTime_: Date, in endTime_: Date, in maxVip_: nat, in maxNormal_: nat, in maxGuest_: nat): FashionShow
+ addEvent(in e: Event): Void
+ cancelEvent(in name: Seq<char>): Void
+ sellTicket(in attendee: Person, in type: TicketType): Ticket
+ inviteGuest(in guest: Person, in presentation: Optional<Presentation>): Void
+ canSellTicket(in type: TicketType): bool
+ getAttendingDesigners(): Set<Person>
+ getAttendingDesignersToEvent(in event: Event): Set<Person>
+ getParticipatingDesigners(): Set<Person>
+ getWorkers(): Set<Person>
+ setNEntries(in type: TicketType, in n: nat): Void
+ setTheme(in theme_: Seq<char>): Void
+ setSpace(in newSpace: Space): Void
+ setTime(in start_: Date, in end_: Date): Void
+ othersINV(in theme: Seq<char>, in startTime: Date, in endTime: Date): bool
+ ticketsINV(in tickets: Map<Ticket::TicketType,Set<Ticket>>, in entries: Map<Ticket::TicketType,nat1>, in theme: Seq<char>): bool
+ eventsINV(in events: Set<Event>, in space: Space, in startTime: Date, in endTime: Date): bool
+ showsOverlap(in s1: FashionShow, in s2: FashionShow): bool
+ equals(in s1: FashionShow, in s2: FashionShow): bool

Image 5 - Class FashionShow

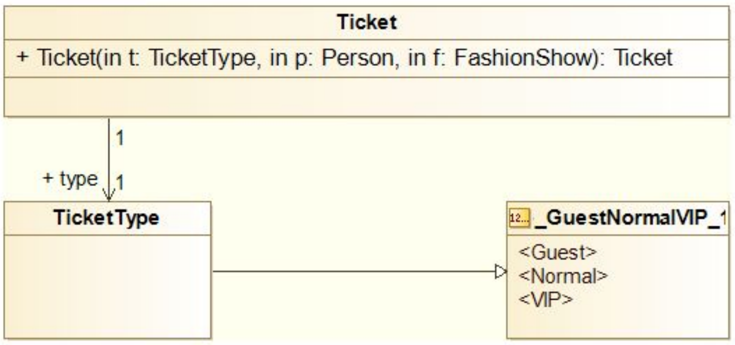
Ticket
+ Ticket(in t: TicketType, in p: Person, in f: FashionShow): Ticket
1
+ type
1
TicketType
_GuestNormalVIP_1
<Guest>
<Normal>
<VIP>

Image 6 - Class Ticket. In the VDM++ the class TicketTypel was defined inside the class Runway as a type.

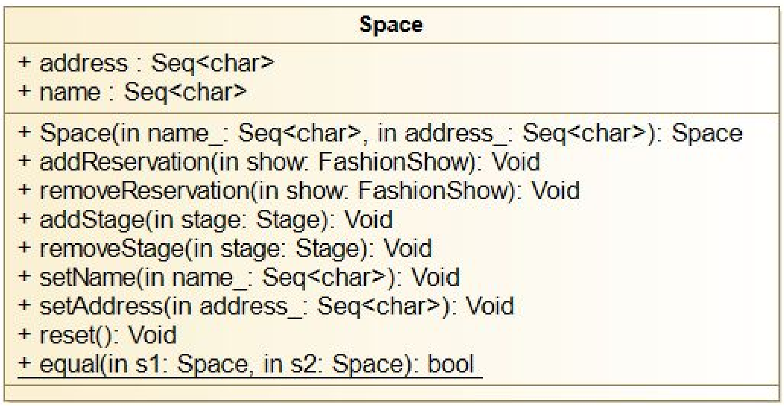
Space
+ address : Seq<char>
+ name : Seq<char>
+ Space(in name_: Seq<char>, in address_: Seq<char>): Space
+ addReservation(in show: FashionShow): Void
+ removeReservation(in show: FashionShow): Void
+ addStage(in stage: Stage): Void
+ removeStage(in stage: Stage): Void
+ setName(in name_: Seq<char>): Void
+ setAddress(in address_: Seq<char>): Void
+ reset(): Void
+ equal(in s1: Space, in s2: Space): bool

Image 7 - Class Space

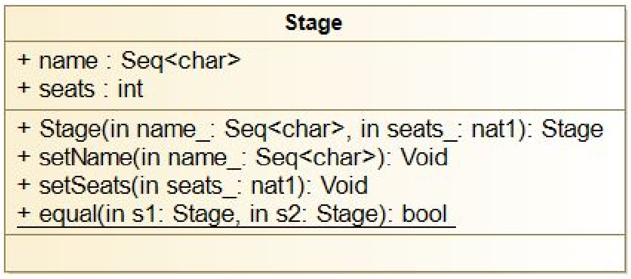
Stage
+ name : Seq<char>
+ seats : int
+ Stage(in name_: Seq<char>, in seats_: nat1): Stage
+ setName(in name_: Seq<char>): Void
+ setSeats(in seats_: nat1): Void
+ equal(in s1: Stage, in s2: Stage): bool

Image 8 - Class Stage

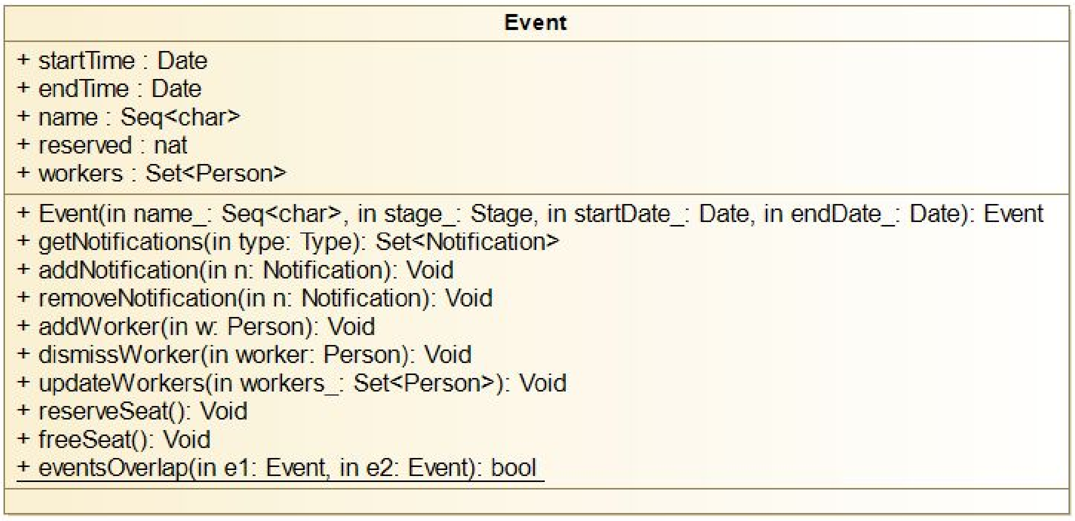
Event
+ startTime : Date
+ endTime : Date
+ name : Seq<char>
+ reserved : nat
+ workers : Set<Person>
+ Event(in name_: Seq<char>, in stage_: Stage, in startDate_: Date, in endDate_: Date): Event
+ getNotifications(in type: Type): Set<Notification>
+ addNotification(in n: Notification): Void
+ removeNotification(in n: Notification): Void
+ addWorker(in w: Person): Void
+ dismissWorker(in worker: Person): Void
+ updateWorkers(in workers_: Set<Person>): Void
+ reserveSeat(): Void
+ freeSeat(): Void
+ eventsOverlap(in e1: Event, in e2: Event): bool

Image 9 - Class Event

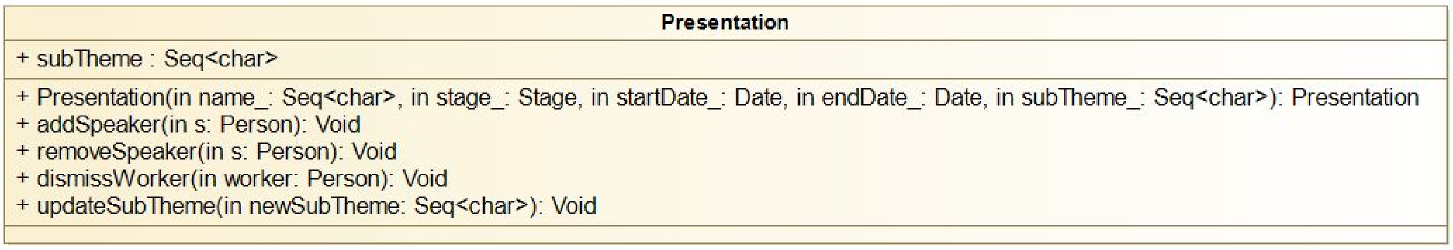
Presentation
+ subTheme : Seq<char>
+ Presentation(in name_: Seq<char>, in stage_: Stage, in startDate_: Date, in endDate_: Date, in subTheme_: Seq<char>): Presentation
+ addSpeaker(in s: Person): Void
+ removeSpeaker(in s: Person): Void
+ dismissWorker(in worker: Person): Void
+ updateSubTheme(in newSubTheme: Seq<char>): Void

Image 10 - Class Presentation

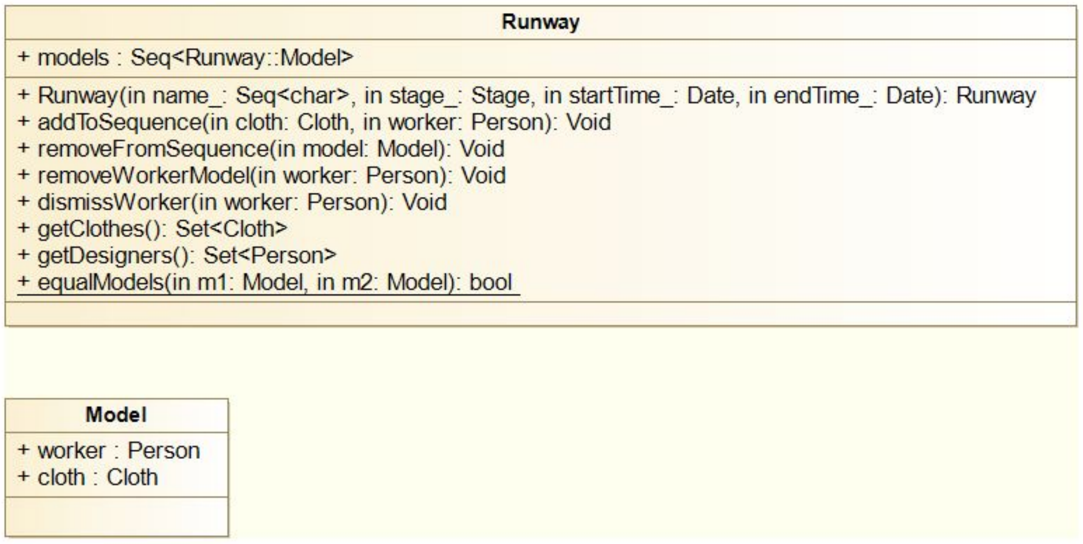
Runway
+ models : Seq<Runway::Model>
+ Runway(in name_: Seq<char>, in stage_: Stage, in startTime_: Date, in endTime_: Date): Runway
+ addToSequence(in cloth: Cloth, in worker: Person): Void
+ removeFromSequence(in model: Model): Void
+ removeWorkerModel(in worker: Person): Void
+ dismissWorker(in worker: Person): Void
+ getClothes(): Set<Cloth>
+ getDesigners(): Set<Person>
+ equalModels(in m1: Model, in m2: Model): bool
Model
+ worker : Person
+ cloth : Cloth

Image 11 - Class Runway. In the VDM++ the class Model was defined inside the class Runway as a type.

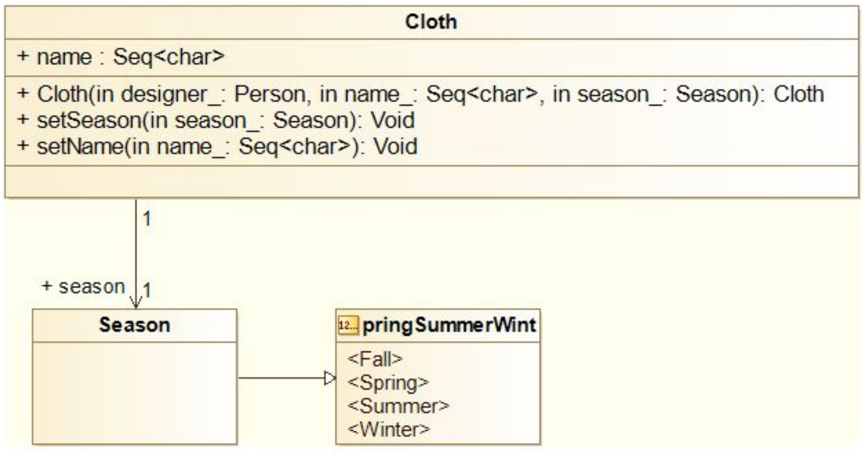
Cloth
+ name : Seq<char>
+ Cloth(in designer_: Person, in name_: Seq<char>, in season_: Season): Cloth
+ setSeason(in season_: Season): Void
+ setName(in name_: Seq<char>): Void
1
+ season
1
Season
pringSummerWint
<Fall>
<Spring>
<Summer>
<Winter>

Image 12 - Class Cloth. In the VDM++ the enumeration was defined inside the class Cloth as a type.

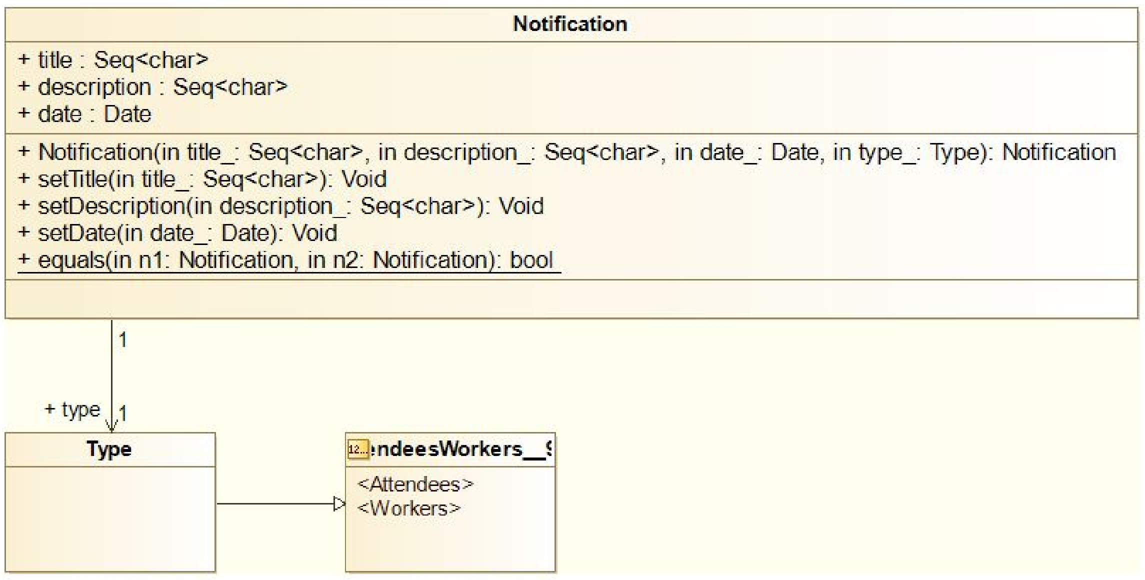
Notification
+ title : Seq<char>
+ description : Seq<char>
+ date : Date
+ Notification(in title_: Seq<char>, in description_: Seq<char>, in date_: Date, in type_: Type): Notification
+ setTitle(in title_: Seq<char>): Void
+ setDescription(in description_: Seq<char>): Void
+ setDate(in date_: Date): Void
+ equals(in n1: Notification, in n2: Notification): bool
1
+ type
1
Type
ndeesWorkers__
<Attendees>
<Workers>

Image 13 - Class Notification. In the VDM++ the enumeration was defined inside the class Notification as a type.

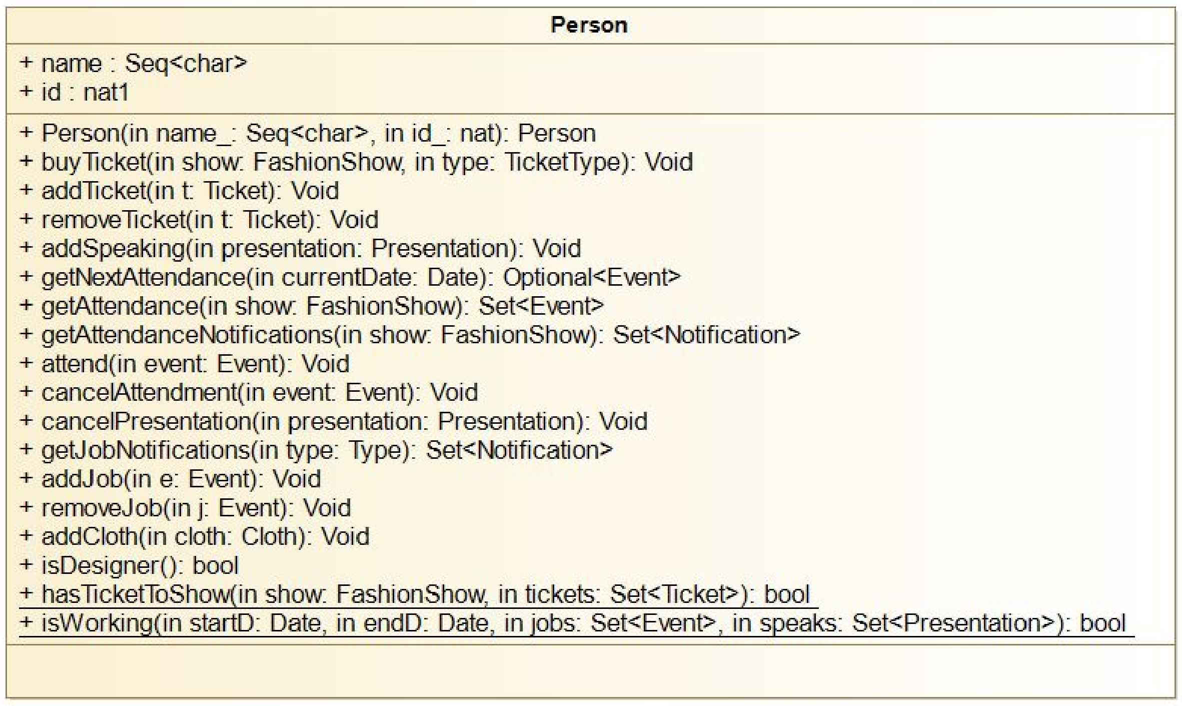
Person
+ name : Seq<char>
+ id : nat1
+ Person(in name_: Seq<char>, in id_: nat): Person
+ buyTicket(in show: FashionShow, in type: TicketType): Void
+ addTicket(in t: Ticket): Void
+ removeTicket(in t: Ticket): Void
+ addSpeaking(in presentation: Presentation): Void
+ getNextAttendance(in currentDate: Date): Optional<Event>
+ getAttendance(in show: FashionShow): Set<Event>
+ getAttendanceNotifications(in show: FashionShow): Set<Notification>
+ attend(in event: Event): Void
+ cancelAttendment(in event: Event): Void
+ cancelPresentation(in presentation: Presentation): Void
+ getJobNotifications(in type: Type): Set<Notification>
+ addJob(in e: Event): Void
+ removeJob(in j: Event): Void
+ addCloth(in cloth: Cloth): Void
+ isDesigner(): bool
+ hasTicketToShow(in show: FashionShow, in tickets: Set<Ticket>): bool
+ isWorking(in startD: Date, in endD: Date, in jobs: Set<Event>, in speaks: Set<Presentation>): bool

Image 14 - Class Person

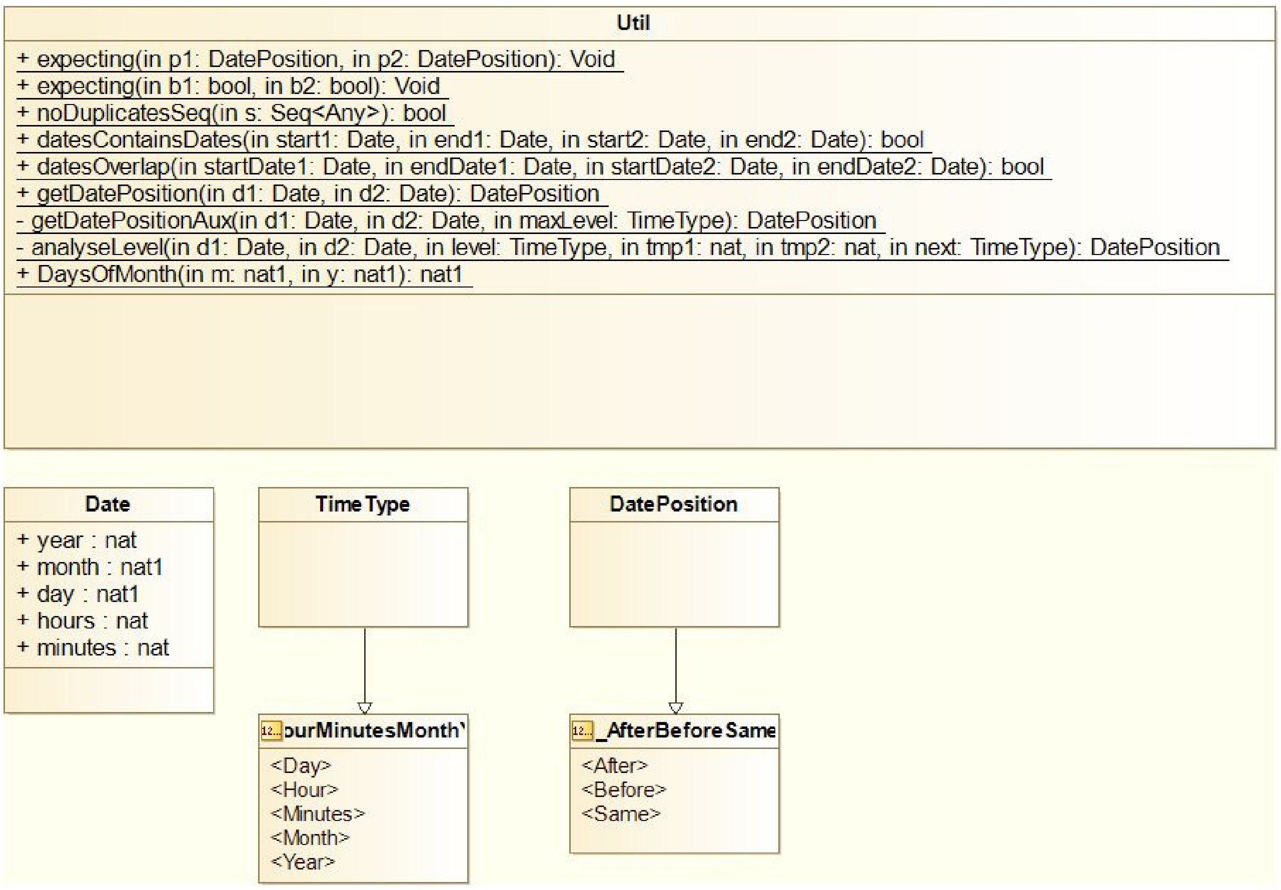
Util
+ expecting(in p1: DatePosition, in p2: DatePosition): Void
+ expecting(in b1: bool, in b2: bool): Void
+ noDuplicatesSeq(in s: Seq<Any>): bool
+ datesContainsDates(in start1: Date, in end1: Date, in start2: Date, in end2: Date): bool
+ datesOverlap(in startDate1: Date, in endDate1: Date, in startDate2: Date, in endDate2: Date): bool
+ getDatePosition(in d1: Date, in d2: Date): DatePosition
- getDatePositionAux(in d1: Date, in d2: Date, in maxLevel: TimeType): DatePosition
- analyseLevel(in d1: Date, in d2: Date, in level: TimeType, in tmp1: nat, in tmp2: nat, in next: TimeType): DatePosition
+ DaysOfMonth(in m: nat1, in y: nat1): nat1
Date
+ year : nat
+ month : nat1
+ day : nat1
+ hours : nat
+ minutes : nat
TimeType
DatePosition
ourMinutesMonth'
<Day>
<Hour>
<Minutes>
<Month>
<Year>
_AfterBeforeSame
<After>
<Before>
<Same>

Image 15 - Utils Class. In VDM++ the class Date and the enumerations TimeType and DatePosition were defined inside the class Utils as types.

Overture

January 3, 2018

Contents

1 Cloth

```
/*
Class to represent a cloth.
*/
class Cloth
types
 public Season = <Winter> | <Fall> | <Summer> | <Spring>;

instance variables
 public designer : Person;
 public name : seq of char;
 public season : Season;

 inv len name > 0;

operations
 /*
Constructor
PRE: name has to be > 0
*/

public Cloth: Person * seq of char * Season ==> Cloth
Cloth(designer_,name_, season_) ==
(designer := designer_; name := name_; season := season_; return self)
pre len name_ > 0;

/*
Set Season
INFO: Sets the season of the cloth
POS: season was set
*/

public setSeason: Season ==> ()
setSeason(season_) == (season := season_)
post season = season_;

/*
Set Name
INFO: Sets the name
PRE: New name > 0
POS: Name was set
*/

public setName: seq of char ==> ()
setName(name_) == (name := name_)
pre len name_ > 0
post name = name_;

end Cloth
```

Function or operation	Line	Coverage	Calls
Cloth	20	100.0%	460
setName	40	100.0%	27
setSeason	30	100.0%	27
Cloth.vdmpp		100.0%	514

2 Event

```
/*
Class that represents an event and has methods
to manipulate its variables.
*/
class Event
 instance variables
   public startTime : Util`Date;
   public endTime : Util`Date;
   public name : seq of char;
   public stage: Stage;
   public reserved: nat := 0;
   public workers : set of Person;
   public news: set of Notification;

   --INV: start time before the end time
 inv Util`getDatePosition(startTime,endTime) = <Before>;
 --INV: workers have this job
 inv forall w in set workers & exists e in set w.jobs &
  e.name = name and e.stage = stage and e.startTime = startTime and e.endTime = endTime;
 --INV: can't exist news with the same content
  inv not exists n1,n2 in set news & n1<> n2 and Notification`equals(n1,n2);

operations
  /*
Constructor
*/

public Event: seq of char * Stage * Util`Date * Util`Date ==> Event
Event(name_, stage_, startDate_, endDate_) == (
 atomic(
 name := name_;
 stage := stage_;
 startTime := startDate_;
 endTime := endDate_;
 workers := {};
 news := {});
 return self;
);

/*
Get Notifications
INFO: Gets the notifications targetting a certain type
POS: RESULT has all the notifications for the target type
*/

public getNotifications: Notification`Type ==> set of Notification
getNotifications(type) == (
 dcl notifications : set of Notification := {};

 for all n in set news do (
  if (n.type = type) then notifications := notifications union {n}
 );

 return notifications
)
post forall n in set RESULT & n.type = type;

/*
Add Notification
PRE: Doesn't exists a notification with the same content
POS: New notification was added
*/

public addNotification: Notification ==> ()
addNotification(n) == (
 news := news union {n};
)
pre not exists n2 in set news & Notification`equals(n,n2)
post n in set news;

/*
Remove Notification
PRE: Notification exists
POS: Notification was removed
*/

public removeNotification: Notification ==> ()
removeNotification(n) == (news := news \ {n})
pre n in set news
post n not in set news;

/*
Add Worker
PRE:New worker can't be part of the workers
POS: New worker was added
*/

public addWorker: Person ==> ()
addWorker(w) == (
w.addJob(self);
workers:= workers union {w}
)
pre w not in set workers
post w in set workers;

/*
Dismiss Worker
PRE: worker exist
POS: worker was dismissed
*/

public dismissWorker: Person ==> ()
dismissWorker(worker) == (
 workers := workers \ {worker};
 worker.removeJob(self);
)
pre worker in set workers
post worker not in set workers;

/*
Update Workers
*/

public updateWorkers: set of Person ==> ()
updateWorkers(workers_) == (
 for all w in set workers_ do w.addJob(self);
 workers := workers_;
);

/*
Reserve a Seat
PRE: Must have free seats
POS: Seat was reserved
*/

public reserveSeat: () ==> ()
reserveSeat() == (reserved := reserved + 1)
pre reserved < stage.seats
post reserved = reserved~ + 1;

/*
Free a Seat
PRE: Mush have reserved seats
POS: 1 Seat was free
*/

public freeSeat: () ==> ()
freeSeat() == (reserved := reserved - 1)
pre reserved > 0
post reserved + 1 = reserved~;

functions
 /*
Events Overlap
INFO: Function that returns true if two events have overlapping
dates in the same space
*/

public static eventsOverlap: Event * Event -> bool
eventsOverlap(e1,e2) == (
 if(e1.stage = e2.stage)
  then Util`datesOverlap(e1.startTime,e1.endTime,e2.startTime,e2.endTime)
 else e1.name = e2.name
);

end Event
```

Function or operation	Line	Coverage	Calls
Event	27	100.0%	942
addNotification	61	100.0%	297
addWorker	83	100.0%	605
dismissWorker	96	100.0%	162
eventsOverlap	139	100.0%	2646
freeSeat	128	100.0%	108
getNotifications	44	100.0%	162
removeNotification	73	100.0%	27
reserveSeat	118	100.0%	355
updateWorkers	107	100.0%	27
Event.vdmpp		100.0%	5331

3 FashionShow

```
/*
Class that allows to manage a show
*/
class FashionShow
 ---------------------------------------------------------------------------
 instance variables
  public theme : seq of char;
  public startTime : Util`Date;
  public endTime : Util`Date;
  public space : Space;

  public events : set of (Event);
  --NEntries: number of entries per ticket type
public nEntries : map Ticket`TicketType to nat1 := {<VIP> |-> 100, <Normal> |-> 100, <Guest> |->
    100};
--ticketsSold: tickets sold per ticket type
 public ticketsSold : map Ticket`TicketType to set of (Ticket) := {<VIP> |-> {}, <Normal> |-> {},
      <Guest> |-> {}};

  inv othersINV(theme,startTime,endTime);
  inv ticketsINV(ticketsSold,nEntries,theme);
  inv eventsINV(events,space,startTime,endTime);
  inv <Guest> in set dom ticketsSold and <VIP> in set dom ticketsSold and <Normal> in set dom
      ticketsSold;
  inv <Guest> in set dom nEntries and <VIP> in set dom nEntries and <Normal> in set dom nEntries;

 ---------------------------------------------------------------------------
 operations
  /*
Constructor
POS: Basic information was set
*/

public FashionShow: seq of char * Space * Util`Date * Util`Date * nat * nat * nat ==> FashionShow
FashionShow(theme_, space_, startTime_, endTime_, maxVip_, maxNormal_, maxGuest_) == (
theme := theme_;
space := space_;
startTime := startTime_;
endTime := endTime_;
nEntries := nEntries ++ {<VIP> |-> maxVip_, <Normal> |-> maxNormal_, <Guest> |-> maxGuest_};
events := {};
space.addReservation(self);
return self )
pre othersINV(theme_,startTime_,endTime_)
post self in set space.reservations and theme = theme_
and startTime = startTime_ and endTime = endTime_ and space = space_;

/*
Add an Event
PRE: Event is between the show dates
PRE: Envet is taking place in a stage belonging to the show's space stages
PRE: Event don't overlap with another
POS: Event was added
*/

public addEvent: Event ==> ()
addEvent(e) == (events := events union {e})
pre Util`datesContainsDates(startTime,endTime,e.startTime,e.endTime) and
  e.stage in set space.stages and not exists e2 in set events & Event`eventsOverlap(e,e2)
post e in set events;

/*
Cancel an Event
```

```
INFO: dismisses all the workers of the event and removes the attendance of the reserved
    attendants
PRE: event exists
POS: event was removed
*/

public cancelEvent: seq of char ==> ()
cancelEvent(name) == (
 dcl event: Event := iota x in set events & x.name = name;
 for all w in set event.workers do ( event.dismissWorker(w) );
 for all t in set dunion rng ticketsSold do (t.owner.cancelAttendment(event));
 events := events \ {event};
)
pre exists e in set events & e.name = name
post (not exists e in set events & e.name = name);

/*
Sell Ticket
INFO: Sells a ticket of a type to a person
PRE: Has tickets to sold
PRE: Type of ticket is not Guest
POS: Ticket was sold
*/

public sellTicket: Person * Ticket`TicketType ==> Ticket
sellTicket(attendee, type) == (
 dcl t :Ticket := new Ticket(type,attendee, self);
 ticketsSold := ticketsSold ++ {type |-> (ticketsSold(type) union {t})};
 return t
)
pre nEntries(type) > card ticketsSold(type) and type <> <Guest>
post (card ticketsSold~(type) + 1 = card ticketsSold(type));

/*
Invite a Guest
INFO: Gives a Guest Ticket to a person and invites it (or not) to be the speecher at a
    presentation
PRE: Guest has free time
PRE: Guest doesn't already have a ticket to this show
POST: Ticket was given
*/

public inviteGuest: Person * [Presentation] ==> ()
inviteGuest(guest,presentation) == (
 dcl t : Ticket := new Ticket(<Guest>,guest,self);
 ticketsSold := ticketsSold ++ { <Guest> |-> ticketsSold(<Guest>) union {t}};

 if(presentation <> nil) then
  presentation.addSpeaker(guest);

 guest.addTicket(t);        --Is given a ticket

 if(presentation <> nil) then(
   guest.addSpeaking(presentation); --Is speaking
   guest.attend(presentation);    --Is attending
 )
)
pre (presentation = nil or Person`isWorking(presentation.startTime,presentation.endTime,guest.
    jobs,guest.speaks) = false)
and not guest.hasTicketToShow(self,guest.tickets)
post exists t in set ticketsSold(<Guest>) & t.owner = guest;

/*
Can Sell a Ticket
*/
```

```
public canSellTicket: Ticket`TicketType ==> bool
canSellTicket(type) == (
 return nEntries(type) > card ticketsSold(type)
);

/*
Get Attendind Designers
POS: All the persons in the RESULT are designers and are attending the show
*/

public getAttendingDesigners: () ==> set of Person
getAttendingDesigners() == (
 dcl designers : set of Person := {};
 for all t in set dunion rng ticketsSold do(
  if(card t.owner.designed > 0) then designers := designers union {t.owner}
 );

 return designers;
)
post forall p in set RESULT & card p.designed > 0 and
 exists t in set p.tickets & t.show = self;

/*
Get Attending Designers To Event
INFO: Only of a specific event
POS: All persons in RESULT are designers and are attending the event(arg)
*/

public getAttendingDesignersToEvent: Event ==> set of Person
getAttendingDesignersToEvent(event) == (
  dcl designers : set of Person := {};
   for all t in set dunion rng ticketsSold do(
    if(card t.owner.designed > 0 and event in set t.owner.attendance(self))
    then designers := designers union {t.owner}
   );

 return designers;
)
 post forall p in set RESULT & card p.designed > 0 and
 exists t in set p.tickets & t.show = self and
 event in set p.attendance(self);

/*
Get Participating Designers
INFO: from all events
POS: All in RESULT are designers and are working in a runway of the show
*/

public getParticipatingDesigners: () ==> set of Person
getParticipatingDesigners() == (
 dcl designers : set of Person := {};

 for all e in set events do(
  if(is_Runway(e)) then designers := designers union e.getDesigners()
 );

 return designers
)
post (forall p in set RESULT & card p.designed > 0 and
(exists event in set p.jobs & event in set events and is_Runway(event)));

/*
Get Workers
POS: All workers in RESULT work at an (at least) one event of the show
```

```
*/

public getWorkers: () ==> set of Person
getWorkers() == (
 dcl workers : set of Person := {};
 for all e in set events do (
  workers := workers union e.workers;
 );
 return workers;
)
post forall w in set RESULT & exists e in set events & e in set w.jobs;


--SETS--


/*
Sets the Number of Entries
PRE: number of entries for each type cannot be bellow the number of tickets sold
POS: number of entries was set
*/

public setNEntries: Ticket`TicketType * nat ==> ()
setNEntries(type,n) == (nEntries := nEntries ++ {type|->n})
pre n >= card ticketsSold(type)
post nEntries(type) = n;

/*
Set Theme
PRE: New theme > 0
POS: Theme was set
*/

public setTheme: seq of char ==> ()
setTheme(theme_) == (theme := theme_)
pre len theme_ > 0
post theme = theme_;

/*
Set Space
POS: space was set
*/

public setSpace: Space ==> ()
setSpace(newSpace) == (
 dcl old : Space := space;
  newSpace.addReservation(self);
  space := newSpace;
  old.removeReservation(self)
)
post space = newSpace;

/*
Set Time
PRE: start date is before the end date
PRE: all the events must be between the time limits
POS: times were set
*/

public setTime: Util`Date * Util`Date ==> ()
setTime(start_,end_) == (atomic(startTime := start_; endTime := end_))
pre Util`getDatePosition(start_,end_) = <Before> and
not exists e in set events &
Util`datesContainsDates(start_,end_,e.startTime,e.endTime) = false
post startTime = start_ and endTime = end_;
```

```
--------------------------------------------------------------------------
functions

--INVARIANTS--

/*
Others Invariant
--INV: start time < end time
--INV: theme must have a name
*/

public static othersINV: seq of char * Util`Date * Util`Date -> bool
othersINV(theme,startTime,endTime) == (
  Util`getDatePosition(startTime,endTime) = <Before> and
  len theme > 0
);

/*
Tickets Invariant
--INV: tickets sold must be regarding the show
--INV: tickets can only be sold once to the same person
--INV: n  of tickets sold must be equal or less than the n  entries
*/

public static ticketsINV: map Ticket`TicketType to (set of (Ticket)) * map Ticket`TicketType to
    nat1 * seq of char -> bool
ticketsINV(tickets,entries,theme) == (
 (forall t in set dunion rng tickets & t.show.theme=theme) and
  (not exists t1,t2 in set dunion rng tickets & t1<>t2 and t1.owner=t2.owner)and
  (forall type in set dom entries & card tickets(type) <= entries(type))
);

/*
Events Invariant
--INV: events in stages belonging to the show
--INV: events between the show dates
--INV: events don't overlap
*/

public static eventsINV: set of Event * Space * Util`Date * Util`Date -> bool
eventsINV(events,space,startTime,endTime) == (
 (forall e in set events & e.stage in set space.stages and
  Util`datesContainsDates(startTime,endTime,e.startTime,e.endTime)) and
  (not exists e1,e2 in set events & e1 <> e2 and Event`eventsOverlap(e1,e2))
);

--OTHERS--

/*
Overlapping shows
INFO: Two shows are overlaping if the dates overlap in the same space
*/

public static showsOverlap: FashionShow * FashionShow -> bool
showsOverlap(s1,s2) == (Space`equal(s1.space,s2.space) and
Util`datesOverlap(s1.startTime,s1.endTime,s2.startTime,s2.endTime)
);

/*
Equal
INFO: Two shows are equal if they have the same theme name
*/

  public static equals: FashionShow * FashionShow -> bool
  equals(s1,s2) == s1.theme = s2.theme;

end FashionShow
```

Function or operation	Line	Coverage	Calls
FashionShow	30	100.0%	1006
addEvent	51	100.0%	452
canSellTicket	118	100.0%	720
cancelEvent	63	100.0%	89
equals	297	100.0%	494
eventsINV	275	100.0%	2956
getAttendingDesigners	127	100.0%	27
getAttendingDesignersToEvent	144	100.0%	27
getParticipatingDesigners	163	100.0%	27
getWorkers	180	100.0%	27
inviteGuest	96	100.0%	54
othersINV	250	100.0%	5083
sellTicket	80	100.0%	1440
setNEntries	199	100.0%	108
setSpace	218	100.0%	54
setTheme	209	100.0%	54
setTime	233	100.0%	60
showsOverlap	288	100.0%	2920
ticketsINV	262	100.0%	5366
FashionShow.vdmpp		100.0%	20964

4 Notification

```
/*
Class that represents a notification
*/
class Notification
types
 public Type = <Workers> | <Attendees>
instance variables
 public title : seq of char;
  public description : seq of char;
  public date : Util`Date;
  public type : Type;

operations
 /*
Constructor
*/

public Notification: seq of char * seq of char * Util`Date * Type ==> Notification
Notification(title_,description_,date_,type_) == (
 title := title_;
 description := description_;
 date := date_;
 type := type_;
 return self
);

/*
Set Title
POS: title was set
*/

public setTitle: seq of char ==> ()
setTitle(title_) == (title := title_)
post title = title_;

/*
Set Description
POS: Description was set
*/

public setDescription: seq of char ==> ()
setDescription(description_) == (description := description_)
post description = description_;

/*
Set Date
POS: date was set
*/

public setDate: Util`Date ==> ()
setDate(date_) == (date := date_)
post date = date_;

functions
 /*
Equals
INFO: Two notifications are equal if the content is the same (title, description,type)
*/

public static equals: Notification * Notification -> bool
equals(n1,n2) == (n1.title = n2.title and n1.description = n2.description and n1.type = n2.type);
end Notification
```

Function or operation	Line	Coverage	Calls
Notification	17	100.0%	567
equals	55	100.0%	864
setDate	46	100.0%	27
setDescription	38	100.0%	27
setTitle	30	100.0%	27
Notification.vdmpp		100.0%	1512

5 Person

```
/*
A Class that represents a person and his
interactivity with the fashion show.
It can play simultanious rolers such as:
a designer, worker and an attendee.
```

```
*/
class Person

instance variables
 public name : seq of char;
 public id : nat1;
 public tickets : set of (Ticket);
  public speaks : set of (Presentation);
  public designed : set of (Cloth);
  public jobs: set of Event;
  public attendance : inmap FashionShow to (set of (Event));

  inv len name > 0;

  --INV: All tickets addressed to the person
 inv forall t in set tickets & t.owner.id = id;

--INV: each presentation (speaks) are being attended in attendance and has guest ticket to the
    show
 inv forall s in set speaks & (
  exists x in set dom attendance & s in set x.events and
  (exists t in set tickets & t.show = x and t.type = <Guest> or t.type = <VIP>))
  or s in set jobs;

 --INV: All clothes designed are marked as designed by the person
 inv forall c in set designed & c.designer.id = id;

 --INV: Attendances only in events belonging to a show with a ticket
  inv forall s in set dom attendance & exists t in set tickets & t.show = s;

 --INV all tickets shows are in attendance
  inv forall t in set tickets & t.show in set dom attendance;
 operations

 /*
 Constructor
 PRE: name > 0
 */

 public Person: seq of char * nat ==> Person
 Person(name_,id_) == (
  name := name_;
  id := id_;
  tickets := {};
  speaks := {};
  designed := {};
  jobs := {};
  attendance := {|->};
  return self)
  pre len name_ > 0;


/*
Buy Ticket
PRE: Don't have ticket for the show
*/

 public buyTicket: FashionShow * Ticket`TicketType ==> ()
 buyTicket(show,type) == (
  if(show.canSellTicket(type))
   then (
   addTicket(show.sellTicket(self,type))
   )
 )
 pre not exists t in set tickets & t.show = show;
 /*
 Add Ticket
 PRE: Person is owner of ticket and don't have ticket to the show
 POS: New Ticket in tickets and show in attendance schedule (per show)
 */

 public addTicket: Ticket ==> ()
 addTicket(t) == (
 atomic(
  tickets := tickets union {t};
  attendance := attendance ++ {t.show |-> {}}
  )
  )
 pre t.owner = self and not exists t2 in set tickets & t.show = t2.show
 post (t in set tickets and t.show in set dom attendance);

/*
REmove Ticket
PRE: Has ticket
POS: Ticket was removed
POS: Is not attendind the show
*/

 public removeTicket: Ticket ==> ()
 removeTicket(t) == (
 for all e in set t.show.events do(
  if(e in set attendance(t.show)) then cancelAttendment(e);
 );
 atomic(
  tickets := tickets \ {t};
  attendance := {t.show} <-: attendance;
  )
 )
 pre t in set tickets
 post t not in set tickets and t.show not in set dom attendance;

  --------------------
----- ATTENDEE -----
--------------------

/*
Add speaking
PRE: Has free time
POS: Speaking was added
*/

public addSpeaking: Presentation ==> ()
addSpeaking(presentation) == (
 speaks := speaks union {presentation};
)
pre (isWorking(presentation.startTime,presentation.endTime,jobs,speaks)) = false or
  presentation in set jobs
post presentation in set speaks;

/*
Get Next Attendance
INFO: Get the next attendance reserved after the current date
*/

public getNextAttendance: Util`Date ==> [Event]
getNextAttendance(currentDate) == (
 dcl event : Event;
 dcl notNull : bool := false;
 for all e in set dunion rng attendance do ( --All events attending
--If it's after the current date
if(Util`getDatePosition(e.startTime,currentDate) = <After>) then(
 --Has to compare with other already found
 if(notNull) then (
  --Compare
    if(Util`getDatePosition(e.startTime, event.startTime) = <After>) then
     event := e;
   )
   else (
    event := e;
    notNull := true;
   )
  )
 );

 return event
);

/*
Get Attendance
INFO: Get the events that have the attendance reserved of a show
PRE: Is attending show
POS: All events attending are in RESULT
*/

public getAttendance: FashionShow ==> set of Event
getAttendance(show) == (
 return attendance(show);
)
pre show in set dom attendance
post forall e in set attendance(show) & e in set RESULT;

/*
Get Attendance Notification
INFO: Get events notifications that have attendance reserved for a show
PRE: show is being attended
POS: notifications in RESULT are for the attendees
*/

 public getAttendanceNotifications: FashionShow ==> set of Notification
 getAttendanceNotifications(show) == (
  dcl notifications : set of Notification := {};

  for all event in set attendance(show) do (
   notifications := notifications union event.getNotifications(<Attendees>)
  );

  return notifications
 )
 pre show in set dom attendance
 post forall n in set RESULT & n.type = <Attendees>;

 /*
Get Attendee Notification
INFO: Notifications from all events that have atendace reserved
POS: notifications in RESULT are for the attendees
*/

 public getAttendeeNotifications: () ==> set of Notification
 getAttendeeNotifications() == (
  dcl notifications : set of Notification := {};

  for all show in set dom attendance do(
   for all event in set attendance(show) do (
    notifications := notifications union event.getNotifications(<Attendees>)
   );
  );

  return notifications
 )
 post forall n in set RESULT & n.type = <Attendees>;


/*
Attend
PRE: Event belongs to a ticket show and is not reserved
POS: Reservation to event with success
*/

 public attend: Event ==> ()
 attend(event) == (
  dcl show: FashionShow := iota x in set dom attendance & event in set x.events,
    ticket: Ticket := iota x in set tickets & show = x.show;
  attendance := attendance ++ {show |-> (attendance(show) union {event})};
  if(ticket.type = <Normal> or ticket.type = <Guest>) then event.reserveSeat()
 )
 pre exists t in set tickets & event in set t.show.events and event not in set attendance(t.show)
 post event in set dunion rng attendance;

/*
Cancel Attendment
POS: Attendace was reserved
*/

 public cancelAttendment: Event ==> ()
 cancelAttendment(event) == (
   if(event in set dunion rng attendance) then (
    dcl show : FashionShow := iota x in set dom attendance & exists e in set attendance(x) & e =
        event;
    attendance := attendance ++ {show |-> (attendance(show) \ {event})};
    event.freeSeat();
  );
  if(is_Presentation(event)) then
   cancelPresentation(event);
 )
 post event not in set dunion rng attendance;

/*
Cancel Presentation (Speach)
PRE: Is Speaker at the presentation
POS: Is no longer the speaker
*/

 public cancelPresentation: Presentation ==> ()
 cancelPresentation(presentation) == (
  speaks := speaks \ {presentation};
  if(self in set presentation.speakers) then
   presentation.removeSpeaker(self);
 )
 pre presentation in set speaks
 post presentation not in set speaks;

 ------------------
 ----- Worker -----
 ------------------

/*
Get Job Notifications
INFO: Get notifications for all the jobs
```

```
POS: Notifications in Result must be
*/

 public getJobNotifications: () ==> set of Notification
 getJobNotifications() == (
  dcl notifications : set of Notification := {};

  for all event in set jobs do(
   notifications := notifications union event.getNotifications(<Workers>)
  );
  return notifications
 )
 post forall n in set RESULT & n.type = <Workers>;

/*
Add Job
PRE: Job does not overlap another
POS: New Job was added to workSchedule
*/

 public addJob: Event ==> ()
 addJob(e) == (jobs := jobs union {e})
 pre isWorking(e.startTime,e.endTime,jobs,speaks) = false
 post e in set jobs;

/*
Remove Job
PRE: Has job
POS: Job was removed
*/

 public removeJob: Event ==> ()
 removeJob(j) ==(jobs := jobs \ {j})
 pre j in set jobs
 post j not in set jobs;

 --------------------
 ----- DESIGNER -----
 --------------------

/*
Add Cloth
PRE: cloth was designed by person and is not in the designed
POS: cloth was added to the designed
*/

 public addCloth: Cloth ==> ()
 addCloth(cloth) == (designed := designed union {cloth})
 pre cloth.designer.id = id and cloth not in set designed
 post cloth in set designed;

/*
Is Designer
*/

public isDesigner: () ==> bool
isDesigner() ==  (
 if(card designed = 0)
  then return false
 else
  return true);

/*
Is Participant Designer
*/
```

```
public isParticipantDesigner: () ==> set of Runway
isParticipantDesigner() ==  (
  dcl participant: set of Runway := {};

  for all event in set jobs do (
  if (is_Runway(event)) then(
   for all i in set inds event.models do(
    if(event.models(i).cloth.designer.id = id) then participant := participant union {event};
   );
  )
  );

  return participant;
);

functions

/*
Has Ticket To Show
*/

 public hasTicketToShow : FashionShow * set of Ticket -> bool
 hasTicketToShow(show,tickets) == (exists t in set tickets & t.show = show);

/*
IS Working
INFO: True is is occupied and false otherwise
*/

public static isWorking: Util`Date * Util`Date * set of Event * set of Presentation-> bool
isWorking(startD,endD,jobs,speaks) == (
(exists event in set jobs & Util`datesOverlap(event.startTime,event.endTime,startD,endD)) or   --
     Has work ?
(exists speak in set speaks & Util`datesOverlap(speak.startTime,speak.endTime,startD,endD))   --
    Has presentation ?
)
pre Util`getDatePosition(startD,endD) = <Before>; --PRE: start time < end time
end Person
```

Function or operation	Line	Coverage	Calls
Person	43	100.0%	59
addCloth	294	100.0%	18
addJob	270	100.0%	37
addSpeaking	112	100.0%	5
addTicket	74	100.0%	37
attend	205	100.0%	13
buyTicket	60	100.0%	99
cancelAttendment	219	94.7%	3
cancelPresentation	236	100.0%	2
getAttendance	154	0.0%	0
getAttendanceNotifications	167	100.0%	2
getAttendeeNotifications	185	100.0%	1
getJobNotifications	254	100.0%	1
getNextAttendance	124	2.8%	0
hasTicketToShow	332	33.3%	0

isDesigner	302	100.0%	1
isParticipantDesigner	312	100.0%	1
isWorking	339	70.8%	0
removeJob	280	100.0%	6
removeTicket	90	76.3%	1
Person.vdmpp		86.5%	286

6 Presentation

```
/*
Class that represents a presentation (Event)
*/
class Presentation is subclass of Event
instance variables
  public speakers: set of (Person);
  public subTheme: seq of char;

operations
 /*
Constructor
*/

public Presentation: seq of char * Stage * Util`Date * Util`Date * seq of char ==> Presentation
Presentation(name_, stage_, startDate_, endDate_, subTheme_) == (
 atomic(
  name := name_;
  stage := stage_;
  startTime := startDate_;
  endTime := endDate_;
  subTheme := subTheme_;
  speakers := {};
  workers := {};
  news := {};
 );
 return self;
);

/*
Add Speaker
PRE:speaker doesn't exist
POS: speaker was added
*/

public addSpeaker: Person ==> ()
addSpeaker(s) ==
(
 if(s not in set workers) then addWorker(s);
 speakers := speakers union {s};
 s.addSpeaking(self);
)
pre s not in set speakers
post s in set speakers;

/*
Remove Speaker
PRE: event not in the speaks of the speaker
PRE: speaker exists
POS: speaker was removed
*/
```

```
public removeSpeaker: Person ==> ()
removeSpeaker(s) == (
 speakers := speakers \ {s};
 if(s not in set workers) then  --If it's a guest
  s.cancelPresentation(self);
)
pre self in set s.speaks or s in set workers
post s not in set speakers;

/*
Dismiss Worker
INFO: Override
PRE: Has worker
POS: Worker was dismissed
*/

public dismissWorker: Person ==> ()
 dismissWorker(worker) == (
  workers := workers \ {worker};
  worker.removeJob(self);
  if(worker in set speakers) then (
   removeSpeaker(worker);
  )
 )
 pre worker in set workers
 post worker not in set workers;

/*
Update subtheme
POS: SubTheme was set
*/

public updateSubTheme: seq of char ==> ()
updateSubTheme(newSubTheme) == (
 subTheme := newSubTheme;
)
post subTheme = newSubTheme;

end Presentation
```

Function or operation	Line	Coverage	Calls
Presentation	13	100.0%	60
addSpeaker	33	100.0%	4
dismissWorker	64	0.0%	0
removeSpeaker	49	91.3%	2
updateSubTheme	79	100.0%	1
Presentation.vdmpp		74.4%	67

7 Runway

```
/*
Class that represents a Runway(Event)
*/
class Runway is subclass of Event
```

```
types
 public Model :: worker : Person
         cloth : Cloth;
instance variables
 public models : seq of (Model);

 --INV: Models are workers and designers of clothes are workers
 inv forall i in set inds models & models(i).worker in set workers
 and models(i).cloth.designer in set workers;


 --INV: Can't have duplicated models
 inv not exists i1,i2 in set inds models &
 i1<>i2 and equalModels(models(i1),models(i2));

operations
/*
Constructor
*/

public Runway: seq of char * Stage * Util`Date * Util`Date ==> Runway
Runway(name_, stage_, startTime_, endTime_) == (
 atomic(
  name := name_;
  stage := stage_;
  startTime := startTime_;
  endTime := endTime_;
  models := [];
  workers := {};
  news := {};
 );
 return self
);

/*
Add Model to Sequece
PRE: model must be a worker at an event
PRE:Can't have duplicated models
POS: Model was added to the sequence
*/

public addToSequence: Cloth * Person ==> ()
addToSequence(cloth,worker) == (
 models := models ^ [mk_Model(worker,cloth)];
)
pre worker in set workers and not exists i in set inds models &
equalModels(models(i),mk_Model(worker,cloth))
post exists i in set inds models & equalModels(models(i),mk_Model(worker,cloth));

/*
Remove from sequece
POS: model was removed
*/

public removeFromSequence: Model ==> ()
removeFromSequence(model) == (
dcl models_ : seq of Model := [];

 for all i in set inds models do(
  if(equalModels(models(i),model) = false) then models_ := models_ ^ [models(i)];
 );

 models := models_;
)
post not exists i in set inds models & equalModels(models(i),model);
```

```
/*
Remove Worker Model
POS: Worker is no longer a model
*/

public removeWorkerModel: Person ==> ()
removeWorkerModel(worker) == (
 dcl models_ : seq of Model := [];

 for all i in set inds models do(
  if(models(i).worker <> worker) then models_ := models_ ^ [models(i)];
 );

 models := models_;
)
post not exists i in set inds models & models(i).worker = worker;

/*
Dismiss Worker
INFO: Override
PRE: Has worker
PRO: Worker was dismiss
*/

public dismissWorker: Person ==> ()
dismissWorker(worker) == (
 if(exists i in set inds models & models(i).worker = worker)
 then removeWorkerModel(worker);
 workers := workers \ {worker};
 worker.removeJob(self);
)
pre worker in set workers
post worker not in set workers and not exists i in set inds models & models(i).worker = worker;

/*
Get Clothes
POS: All in Result are clothes used by the models
*/

public getClothes: () ==> set of Cloth
getClothes() == (
 dcl clothes : set of Cloth := {};
 for all i in set inds models do(
  if(models(i).cloth not in set clothes) then clothes := clothes union {models(i).cloth}
 );
 return clothes
)
post forall i in set inds models & models(i).cloth in set RESULT;

/*
Get Designers
POS: All in RESULT are designers and are working at the event
*/

public getDesigners: () ==> set of Person
getDesigners() == (
 dcl designers : set of Person := {};
 for all i in set inds models do(
  if(models(i).cloth.designer not in set designers) then designers := designers union {models(i).
      cloth.designer}
 );
 return designers
)
post forall des in set RESULT & card des.designed > 0 and exists e in set des.jobs & e = self;
```

```
functions
/*
Equal Models
*/

public equalModels: Model * Model -> bool
equalModels(m1,m2) == (m1.cloth = m2.cloth and m1.worker = m2.worker);
end Runway
```

Function or operation	Line	Coverage	Calls
Runway	24	100.0%	139
addToSequence	44	100.0%	239
dismissWorker	90	100.0%	32
equalModels	133	100.0%	1534
getClothes	104	100.0%	96
getDesigners	118	100.0%	70
removeFromSequence	56	100.0%	16
removeWorkerModel	72	100.0%	32
Runway.vdmpp		100.0%	2158

8 ShowsManager

```
/*
Class that allows to manage multiple shows
*/
class ShowsManager

instance variables
 public shows: set of (FashionShow) := {};

 --INV: Can't have the same shows with the same theme with overapping dates
 inv not exists s1,s2 in set shows & s1<>s2 and FashionShow`equals(s1,s2) and
 Util`datesOverlap(s1.startTime,s1.endTime,s2.startTime,s2.endTime);

operations
/*
Constructor
*/

public ShowsManager: () ==> ShowsManager
ShowsManager() == (return self);

/*
Add a Show
PRE: show is not being managed and doesn't overlap
POS. show was added
*/

public addShow: FashionShow ==> ()
addShow(fs) == (shows := shows union {fs})
pre fs not in set shows
post fs in set shows;
```

```
/*
Remove Show
PRE: show is being manages
POS: show was removed
*/

public removeShow: FashionShow ==> ()
removeShow(show) == (
dcl space: Space := show.space;
 for all e in set show.events do(
  show.cancelEvent(e.name)
 );
 for all t in set dunion rng show.ticketsSold do(
  t.owner.removeTicket(t);
 );
 shows := shows \ {show}
)
pre show in set shows
post show not in set shows;

/*
Get Spaces
INFO: Get all the spaces used by all the shows that are being managed
POS: All in RESULT are used by one or more show
*/

public getSpaces: () ==> set of Space
getSpaces() == (
 dcl spaces : set of Space := {};
 for all s in set shows do(
  if(s.space not in set spaces) then spaces := spaces union {s.space}
 );
 return spaces
)
post forall s in set RESULT &
exists show in set shows & show in set s.reservations;

/*
Reset
*/

public reset: () ==> ()
reset() == shows := {};

end ShowsManager
```

Function or operation	Line	Coverage	Calls
ShowsManager	17	100.0%	32
addShow	25	100.0%	192
getSpaces	54	100.0%	16
removeShow	35	100.0%	32
reset	68	100.0%	48
ShowsManager.vdmpp		100.0%	320

9 Space

```
/*
Class that represents a space
*/
class Space
 ---------------------------------------------------------------------------
instance variables
  public address : seq of char;
  public name : seq of char;
  public stages : set of Stage;
  public reservations : set of FashionShow;

  --INV: address and name > 0 and stages are different
  inv len address > 0 and len name > 0 and not exists s1,s2 in set stages &
  s1<>s2 and Stage`equal(s1,s2);

  --INV: reservations can't overlap
  inv not exists s1,s2 in set reservations &
  s1<>s2 and FashionShow`showsOverlap(s1,s2);
 ---------------------------------------------------------------------------
operations
/*
Constructor
PRE: name and address are > 0
POS: name and address were set
*/

public Space: seq of char * seq of char ==> Space
Space(name_,address_) == (
 name := name_ ;
 address := address_ ;
 stages := {};
 reservations := {};
 return self)
pre len name_ > 0 and len address_ > 0
post name = name_ and address = address_;

/*
Add Reservation
PRE: Reservations can't overlap
POS: Reservation with success
*/

public addReservation: FashionShow ==> ()
addReservation(show) == (reservations := reservations union {show})
pre forall s in set reservations & FashionShow`showsOverlap(s,show)=false
post show in set reservations;

/*
Remove Reservation
PRE: Show has reservation
POS: Reservation cancelled
*/

public removeReservation: FashionShow ==> ()
removeReservation(show) == (reservations := reservations \ {show})
pre show in set reservations
post show not in set reservations;

/*
Add Stage
PRE: Stage not in stages
PRE: Doesn't exist a stage with the same name
POS: Stage was added
*/
```

```
public addStage: Stage ==> ()
addStage(stage) == (stages := stages union {stage})
pre stage not in set stages and not exists s in set stages & Stage`equal(s,stage)
post stage in set stages;

/*
Remove Stage
PRE: Has Stage
POS: stage was removed
*/

public removeStage: Stage ==> ()
removeStage(stage) == (
 for all s in set reservations do (
   for all e in set s.events do (
    if(Stage`equal(e.stage,stage)) then s.cancelEvent(e.name);
   );
 );
 stages := stages \ {stage};
)
pre stage in set stages
post stage not in set stages;

/*
Set name
PRE: name > 0
POS: name changed
*/

public setName: seq of char ==> ()
setName(name_) == (name := name_)
pre len name_ > 0
post name = name_;

/*
Set Address
PRE: address > 0
POS:address changed
*/

public setAddress: seq of char ==> ()
setAddress(address_) == (address := address_)
pre len address_ > 0
post address = address_;

/*
Reset
*/

public reset: () ==> ()
reset() == (reservations := {})
post reservations = {};
 ---------------------------------------------------------------------------
 functions

/*
Equal
*/

public static equal: Space * Space -> bool
equal(s1,s2) == (s1.address = s2.address);
end Space
```

Function or operation	Line	Coverage	Calls
Space	26	100.0%	1000
addReservation	41	100.0%	1060
addStage	62	100.0%	524
equal	116	100.0%	3942
removeReservation	51	100.0%	54
removeStage	72	100.0%	43
reset	107	100.0%	16
setAddress	99	100.0%	16
setName	89	100.0%	16
Space.vdmpp		100.0%	6671

10 Stage

```
/*
Class that represents a stage
*/
class Stage
 ---------------------------------------------------------------------------
instance variables
 public name : seq of char;
 public seats : int;

 --INV
 inv seats > 0 and len name > 0;
 ---------------------------------------------------------------------------
operations
/*
Constructor
PRE: name > 0
POS: basic infomation was set
*/

public Stage: seq of char * nat1 ==> Stage
Stage(name_,seats_) == (name := name_; seats := seats_; return self)
pre len name_ > 0
post name = name_ and seats = seats_;

/*
Set Name
PRE: name > 0
POS: name was set
*/

public setName: seq of char ==> ()
setName(name_) == (name := name_)
pre len name_ > 0
post name = name_;

/*
Set Seats
PRE: can only add seats
POS: seats was added
*/

public setSeats: nat1 ==> ()
setSeats(seats_) == (seats := seats_)
```

```
pre seats < seats_
post seats = seats_;
 ---------------------------------------------------------------------------
functions

/*
Equal
*/

public static equal: Stage * Stage -> bool
equal(s1,s2) == (s1.name = s2.name);

end Stage
```

Function or operation	Line	Coverage	Calls
Stage	19	100.0%	994
equal	49	100.0%	6574
setName	29	100.0%	32
setSeats	39	100.0%	32
Stage.vdmpp		100.0%	7632

11 Ticket

```
/*
Class that represents a ticket
*/
class Ticket
types
 public static TicketType = <Normal> | <VIP> | <Guest>;
instance variables
  public type : TicketType;
  public owner : Person;
  public show : FashionShow;

operations
 /*
 Constructor
 */

 public Ticket: TicketType * Person * FashionShow ==> Ticket
 Ticket(t,p,f) == (atomic(type := t; owner := p; show := f); return self);
end Ticket
```

Function or operation	Line	Coverage	Calls
Ticket	16	100.0%	828
Ticket.vdmpp		100.0%	828

12 Util

```
/*
Class with util methods and types.
*/
class Util
types
 public static TimeType = <Year> | <Month> | <Day> | <Hour> | <Minutes>;
 public static DatePosition = <Before> | <Same> | <After>;
 public Date :: year : nat
         month: nat1
         day: nat1
         hours: nat
         minutes: nat
         inv d == d.month <= 12 and d.day <= DaysOfMonth(d.month,d.year) and d.hours < 24 and d.
             minutes < 60;

operations
/*
TESTS
*/

/*
Expecting
PRE: arguments must be equals
*/

 public static expecting: Util`DatePosition * Util`DatePosition ==> ()
 expecting(p1, p2) == return
 pre p1 = p2;

 /*
 expecting
 PRE: arguments must be equals
 */
 public static expecting: bool * bool ==> ()
 expecting(b1, b2) == return
 pre b1 = b2;

functions

 /*
DATE
*/
/*
Dates Contains Dates
INFO: Method that retuns true if a (start1,end1) date contains (start2,end2) date fully
*/

public static datesContainsDates: Date * Date * Date * Date -> bool
datesContainsDates(start1,end1,start2,end2) == (
 getDatePosition(start1,start2) <> <After> and getDatePosition(end1,end2) <> <Before>
);

/*
Dates Overlap
INFO: Method that returns true if the dates overlap and false otherwise
*/

public static datesOverlap: Date * Date * Date * Date -> bool
datesOverlap(startDate1, endDate1, startDate2, endDate2) ==
(
 let o1 = getDatePosition(startDate1,startDate2), --StartDate1 vs StartDate2
 o2 = getDatePosition(startDate1,endDate2),  --StartDate1 vs EndDate2
 o3 = getDatePosition(endDate1,startDate2),  --EndDate1 vs StartDate2
 o4 = getDatePosition(endDate1,endDate2)  --EndDate1 vs EndDate2
 in (
 if((o1 = <Before> or o1 = <Same>) and (o3 = <Same> or o3 = <Before>)) or
  ((o2 = <Same> or o2 = <After>) and (o4 = <Same> or o4 = <After>)))
  then false
 else
  true
 )
);

/*
Get Date Position
INFO: Method that returns the first date position regarding the second (in the parameters)
*/

public static getDatePosition: Date * Date -> DatePosition
getDatePosition(d1, d2) == getDatePositionAux(d1,d2,<Year>);


private static getDatePositionAux: Date * Date * TimeType -> DatePosition
getDatePositionAux(d1, d2, maxLevel) == (

 cases maxLevel:
  <Year> -> analyseLevel(d1,d2,maxLevel,d1.year,d2.year,<Month>),
  <Month> -> analyseLevel(d1,d2,maxLevel,d1.month,d2.month,<Day>),
  <Day> -> analyseLevel(d1,d2,maxLevel,d1.day,d2.day,<Hour>),
  <Hour> -> analyseLevel(d1,d2,maxLevel,d1.hours,d2.hours,<Minutes>),
  <Minutes> -> analyseLevel(d1,d2,maxLevel,d1.minutes,d2.minutes,<Minutes>)
  end
);

/*
Analyse Level
INFO: Auxiliary Function
*/

private static analyseLevel: Date * Date * TimeType * nat * nat * TimeType -> DatePosition
analyseLevel(d1,d2,level,tmp1,tmp2,next) ==
if(tmp1 = tmp2)
 then (
  if(level <> <Minutes>)
   then getDatePositionAux(d1,d2,next)
  else
   <Same>
 )
 else if(tmp1 > tmp2)
 then <After>
 else
 <Before>;

/*
Days of the Month
INFO: Method that returns the days of the month
*/

public static DaysOfMonth: nat1 * nat1 +> nat1
DaysOfMonth(m, y) ==
(cases m:
  1,3,5,7,8,10,12 -> 31,
  4,5,9,11 -> 30,
  2 -> (if((y mod 4 = 0 and y mod 100 <> 0) or (y mod 400 = 0)) then 29 --Leap Year
      else 28)
  end);
end Util
```

Function or operation	Line	Coverage	Calls
DaysOfMonth	111	100.0%	29
analyseLevel	93	100.0%	88980
datesContainsDates	45	100.0%	3468
datesOverlap	54	100.0%	4162
expecting	24	100.0%	63
getDatePosition	74	100.0%	24231
getDatePositionAux	77	100.0%	3840
Util.vdmpp		100.0%	124773

13 TestAll

```
class TestAll
operations

public main: () ==> ()
main() == (
 new TestCloth().main();
 new TestEvent().main();
 new TestFashionShow().main();
 new TestNotification().main();
 new TestPerson().main();
 new TestPresentation().main();
 new TestRunway().main();
 new TestShowsManager().main();
 new TestSpace().main();
 new TestStage().main();
 new TestUtil().main();
);
end TestAll
```

Function or operation	Line	Coverage	Calls
main	3	100.0%	144
TestAll.vdmpp		100.0%	144

14 TestClass

```
class TestClass
operations

  protected expecting: ? * ? ==> ()
  expecting(comp1, comp2) == return
  pre comp1 = comp2;


  protected expectingIn: ? * set of ? ==> ()
  expectingIn(comp1, comp2) == return
  pre comp1 in set comp2;
  protected expectingNotIn: ? * set of ? ==> ()
  expectingNotIn(comp1, comp2) == return
  pre comp1 not in set comp2;

end TestClass
```

Function or operation	Line	Coverage	Calls
expecting	3	100.0%	6037
expectingIn	7	100.0%	1024
expectingNotIn	11	100.0%	572
TestClass.vdmpp		100.0%	7633

15 TestCloth

```
class TestCloth is subclass of TestClass
operations

  private testSetSeason: () ==> ()
  testSetSeason() == (
   dcl designer: Person := new Person("Designer", 1);
   dcl cloth: Cloth := new Cloth(designer, "Coat", <Winter>);

   cloth.setSeason(<Fall>);
   expecting(cloth.season, <Fall>);
  );


  private testSetName: () ==> ()
  testSetName() == (
   dcl designer: Person := new Person("Designer", 1);
   dcl cloth: Cloth := new Cloth(designer, "Coat", <Winter>);

   cloth.setName("Fedora");
   expecting(cloth.name, "Fedora");

   -- cloth.setName(""); --> BREAKS (name len = 0)
  );


  public static main: () ==> ()
 main() == (
   dcl t: TestCloth := new TestCloth();
   t.testSetSeason();
   t.testSetName();
  );

end TestCloth
```

Function or operation	Line	Coverage	Calls
main	23	100.0%	87
testSetName	12	100.0%	142

testSetSeason	3	100.0%	142
TestCloth.vdmpp		100.0%	371

16 TestEvent

```
class TestEvent is subclass of TestClass
operations
  -- tests requirement R19

 private testAddWorker: () ==> ()
  testAddWorker() == (
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,20,55);
   dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
   dcl stage: Stage := new Stage("B001", 200);
   dcl event: Event := new Event("Keynote", stage, startTime, endTime);

   dcl worker: Person := new Person("Person1", 111111111);

   event.addWorker(worker);
   -- event.addWorker(worker); --> BREAKS (can't add same worker twice)

   expecting(event.workers, {worker});
  );

  -- tests requirement R19

 private testDismissWorker: () ==> ()
  testDismissWorker() == (
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,20,55);
   dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
   dcl stage: Stage := new Stage("B001", 200);
   dcl event: Event := new Event("Keynote", stage, startTime, endTime);

   dcl worker: Person := new Person("Person1", 111111111);

   event.addWorker(worker);
   event.dismissWorker(worker);

   expecting(event.workers, {});

   -- event.dismissWorker(worker); --> BREAKS (can't dismiss a person that is not working)
  );


  private testGetNotifications: () ==> ()
  testGetNotifications() == (
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,20,55);
   dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
   dcl stage: Stage := new Stage("B001", 200);
   dcl event: Event := new Event("Keynote", stage, startTime, endTime);

   dcl notificationTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
   dcl notification1: Notification := new Notification("title", "Keynote to start",
       notificationTime, <Workers>);
   dcl notification2: Notification := new Notification("title", "Keynote ended", notificationTime
       , <Attendees>);

  dcl workerNotifications: set of Notification;
  dcl attendeeNotifications: set of Notification;
```

```
event.news := {notification1, notification2};
workerNotifications := event.getNotifications(<Workers>);
attendeeNotifications := event.getNotifications(<Attendees>);

expecting(workerNotifications, {notification1});
expecting(attendeeNotifications, {notification2});
);

-- tests requirement 25

private testAddNotification: () ==> ()
testAddNotification() == (
dcl startTime: Util`Date := mk_Util`Date(1997,02,14,20,55);
 dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
 dcl stage: Stage := new Stage("B001", 200);
 dcl event: Event := new Event("Keynote", stage, startTime, endTime);

 dcl notificationTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
 dcl notification1: Notification := new Notification("title", "Keynote started",
     notificationTime, <Attendees>);
 dcl notification2: Notification := new Notification("title", "Keynote ended", notificationTime
     , <Attendees>);

 event.addNotification(notification1);
 expecting(event.news, {notification1});

 event.addNotification(notification2);
 expecting(event.news, {notification1, notification2});

 -- event.addNotification(notification2); --> BREAKS (can't add same notification twice)
);


private testRemoveNotification: () ==> ()
testRemoveNotification() == (
dcl startTime: Util`Date := mk_Util`Date(1997,02,14,20,55);
 dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
 dcl stage: Stage := new Stage("B001", 200);
 dcl event: Event := new Event("Keynote", stage, startTime, endTime);

 dcl notificationTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
 dcl notification: Notification := new Notification("title", "Keynote started",
     notificationTime, <Attendees>);

 event.addNotification(notification);
 event.removeNotification(notification);
 expectingNotIn(notification, event.news);

 -- event.removeNotification(notification); --> BREAKS (can't remove a notification that doesn'
     t exist)
);


private testUpdateWorkers: () ==> ()
testUpdateWorkers() == (
dcl startTime: Util`Date := mk_Util`Date(1997,02,14,20,55);
 dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
 dcl stage: Stage := new Stage("B001", 200);
 dcl event: Event := new Event("Keynote", stage, startTime, endTime);

 dcl worker1: Person := new Person("Person1", 1);
 dcl worker2: Person := new Person("Person2", 2);

 dcl workerSet: set of Person := {worker1, worker2};
 event.updateWorkers(workerSet);
```

```
    expecting(event.workers, workerSet);
   );


  private testReserveSeat: () ==> ()
   testReserveSeat() == (
   dcl startTime: Util`Date := mk_Util`Date(1997,02,14,20,55);
    dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
    dcl stage: Stage := new Stage("B001", 1);
    dcl event: Event := new Event("Keynote", stage, startTime, endTime);

    event.reserveSeat();
    -- event.reserveSeat(); --> BREAKS (can't reserve seat in a full stage);

    expecting(event.reserved, 1);
   );


  private testFreeSeat: () ==> ()
   testFreeSeat() == (
   dcl startTime: Util`Date := mk_Util`Date(1997,02,14,20,55);
    dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
    dcl stage: Stage := new Stage("B001", 1);
    dcl event: Event := new Event("Keynote", stage, startTime, endTime);

    event.reserveSeat();
    event.freeSeat();
    expecting(event.reserved, 0);
    -- event.freeSeat(); --> BREAKS (can't free seat in an empty stage);
   );


   private testEventsOverlap: () ==> ()
   testEventsOverlap() == (
   dcl startTime1: Util`Date := mk_Util`Date(1997,02,14,20,55);
    dcl endTime1: Util`Date := mk_Util`Date(1997,02,14,21,55);
   dcl startTime2: Util`Date := mk_Util`Date(1997,02,14,21,25);
    dcl endTime2: Util`Date := mk_Util`Date(1997,02,14,22,25);
    dcl stage1: Stage := new Stage("B001", 200);
    dcl stage2: Stage := new Stage("B002", 200);
    dcl event1: Event := new Event("Keynote", stage1, startTime1, endTime1);
    dcl event2: Event := new Event("Ending Keynote", stage1, startTime2, endTime2);
    dcl event3: Event := new Event("Ending Keynote", stage2, startTime2, endTime2);

    dcl overlap : bool := Event`eventsOverlap(event1, event2);
    dcl non_overlap : bool := Event`eventsOverlap(event1, event3);

    expecting(overlap, true); -- same stage
    expecting(non_overlap, false); -- different stage
   );


   public static main: () ==> ()
  main() == (
    dcl t: TestEvent := new TestEvent();
    t.testAddWorker(); -- R19 and R5
    t.testDismissWorker(); -- R19 and R6
    t.testGetNotifications();
    t.testAddNotification(); -- R25
    t.testRemoveNotification();
    t.testUpdateWorkers();
    t.testReserveSeat();
    t.testFreeSeat();
    t.testEventsOverlap();
   );

 end TestEvent
```

Function or operation	Line	Coverage	Calls
main	158	100.0%	86
testAddNotification	60	100.0%	27
testAddWorker	4	100.0%	27
testDismissWorker	20	100.0%	27
testEventsOverlap	139	100.0%	86
testFreeSeat	126	100.0%	28
testGetNotifications	37	100.0%	27
testRemoveNotification	80	100.0%	27
testReserveSeat	113	100.0%	27
testUpdateWorkers	97	100.0%	27
TestEvent.vdmpp		100.0%	389

17 TestFashionShow

```
class TestFashionShow is subclass of TestClass

instance variables
 public show : FashionShow;

operations


 protected assert : bool ==> ()
  assert(a) == return
  pre a;


 public testConstructor:() ==> ()
 testConstructor() == (
  dcl space1 : Space := new Space("name1","address1"),
    space2 : Space := new Space("name2","address2");

  show := new FashionShow("theme",space1,mk_Util`Date(1,1,1,1,1),mk_Util`Date(1,1,1,1,2),10, 10,
      10);
  assert(show.theme = "theme" and show.space = space1 and show.startTime = mk_Util`Date
      (1,1,1,1,1) and
     show.endTime = mk_Util`Date(1,1,1,1,2) and show.nEntries(<Normal>) = 10 and
     show.nEntries(<VIP>) = 10 and show.nEntries(<Guest>) = 10);

  --show := new FashionShow("",space1,mk_Util`Date(1,1,1,1,1),mk_Util`Date(1,1,1,1,2),10, 10, 10)
      ;     -->BREAKS (theme len = 0)
  --show := new FashionShow("theme",space1,mk_Util`Date(1,1,1,1,2),mk_Util`Date(1,1,1,1,1),10,
      10, 10); -->BREAKS (end data before start date)
  --show := new FashionShow("theme",space1,mk_Util`Date(1,1,1,1,1),mk_Util`Date(1,1,1,1,1),10,
      10, 10); -->BREAKS (end data same as start date)

  assert(FashionShow`equals(show,show));
  assert(FashionShow`equals(show,new FashionShow("theme",space2,mk_Util`Date(2,2,2,2,2),mk_Util`
      Date(3,3,3,3,3),20, 30, 40)));      --same name
  assert(FashionShow`equals(show,new FashionShow("theme2",space2,mk_Util`Date(3,3,3,3,3),mk_Util`
      Date(4,4,4,4,4),20, 30, 40)) = false); --different name

  assert(FashionShow`showsOverlap(show,show));
  --assert(FashionShow`showsOverlap(show,new FashionShow("theme2",space1,mk_Util`Date(1,1,1,1,1),
      mk_Util`Date(5,5,5,5,5),20, 30, 40)));-->BREAKS dates overlap on same address
  return
 );


 public testEventsManager: () ==> ()
 testEventsManager() == (
  dcl e1 : Event, e2 : Event, e3 : Event,
    s1 : Space := new Space("name1","address1"),
    st1: Stage , st2: Stage, st3 : Stage,
    p1 : Person, p2 : Person, p3 : Person, p4 : Person;

  show := new FashionShow("theme",new Space("name2","address"),mk_Util`Date(2017,01,01,10,00),
      mk_Util`Date(2017,02,01,10,00),10, 10, 100);

  --sets

  show.setSpace(s1);
  assert(show.space = s1);

  show.setTheme("t");
  assert(show.theme = "t");

  show.setNEntries(<Normal>,20);
  assert(show.nEntries(<Normal>) = 20);

  show.setTime(mk_Util`Date(2017,01,01,08,00),mk_Util`Date(2017,02,02,10,00)); -- Expand time
      limit
  assert(show.startTime =mk_Util`Date(2017,01,01,08,00) and show.endTime = mk_Util`Date
      (2017,02,02,10,00));

  --add Stages

  st1 := new Stage("stage1",100);
  st2 := new Stage("stage2",50);
  st3 := new Stage("stage3",20);

  s1.addStage(st1);
  s1.addStage(st2);
  s1.addStage(st3);
  --s1.addStage(new Stage(1000,"stage1")); -->BREAKS (stage with duplicated name = s1)

  e1 := new Event("name1", st1, mk_Util`Date(2017,01,01,10,00), mk_Util`Date(2017,01,01,16,00));
  e2 := new Event("name2", st2, mk_Util`Date(2017,01,01,16,00), mk_Util`Date(2017,01,01,20,00));
  e3 := new Event("name3", st3, mk_Util`Date(2017,01,02,10,00), mk_Util`Date(2017,01,02,16,00));

  --add Events

  show.addEvent(e1);
  show.addEvent(e2);
  show.addEvent(e3);
  --show.addEvent(new Event("name1", st2, mk_Util`Date(2017,01,01,10,00), mk_Util`Date
      (2017,01,01,16,00))); -->BREAKS (event with same name);
  --show.addEvent(new Event("name4", st1, mk_Util`Date(2017,01,01,12,00), mk_Util`Date
      (2017,01,01,17,00))); -->BREAKS (event in the same space at the same time);
  --show.addEvent(new Event("name5", st2, mk_Util`Date(2016,12,31,23,00), mk_Util`Date
      (2017,01,01,16,00))); -->BREAKS (event outside the show date limits);


  --buy tickets
  p1 := new Person("name1",1);
  p2 := new Person("name2",2);
  p3 := new Person("name3",3);
  p4 := new Person("name4",4);

  p1.buyTicket(show, <Normal>);
  p2.buyTicket(show, <Normal>);
  p1.attend(e1);
  p2.attend(e2);
  e1.addWorker(p3);
  e2.addWorker(p4);

  show.cancelEvent("name1");
  assert(e1 not in set show.events);
  assert(e1 not in set p1.attendance(show));
  assert(e1 not in set p3.jobs);

  show.space.removeStage(st2);
  assert(e2 not in set show.events);
  assert(e2 not in set p2.attendance(show));
  assert(st2 not in set show.space.stages);
  assert(e2 not in set p4.jobs);

  return
 );

  -- tests requirement R18

 private testAddEvent: () ==> ()
 testAddEvent() == (
   dcl space: Space := new Space("name", "address");
   dcl startTimeShow1: Util`Date := mk_Util`Date(1997,02,13,20,55);
   dcl endTimeShow1: Util`Date := mk_Util`Date(1997,02,15,22,00);
   dcl startTime1: Util`Date := mk_Util`Date(1997,02,14,21,00);
   dcl endTime1: Util`Date := mk_Util`Date(1997,02,14,21,55);
   dcl startTime2: Util`Date := mk_Util`Date(1998,02,14,21,00);
   dcl endTime2: Util`Date := mk_Util`Date(1998,02,14,21,55);
   dcl stage: Stage := new Stage("B001", 200);

   dcl event1: Event := new Event("Keynote", stage, startTime1, endTime1);
   dcl event2: Event := new Event("Keynote", stage, startTime2, endTime2);
   dcl event3: Event := new Event("Keynote", stage, startTime1, endTime1);
   dcl show: FashionShow := new FashionShow("theme", space, startTimeShow1, endTimeShow1, 10, 10,
        100);

  space.addStage(stage);
  show.addEvent(event1);
  -- show.addEvent(event2); --> BREAKS (can't add an event that occurs outside the date interval
      of the show)
  -- show.addEvent(event3); --> BREAKS (can't add an event that ovelaps temporally with another
      in the same stage)

  expecting(show.events, {event1});
 );

  -- tests requirement R18

 private testCancelEvent: () ==> ()
 testCancelEvent() == (
   dcl space: Space := new Space("name", "address");
   dcl startTimeShow: Util`Date := mk_Util`Date(1997,02,13,20,55);
   dcl endTimeShow: Util`Date := mk_Util`Date(1997,02,15,22,00);
   dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
   dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
   dcl stage: Stage := new Stage("B001", 200);
   dcl event1: Event := new Event("Keynote", stage, startTime, endTime);
   dcl event2: Event := new Event("ClosingKeynote", stage, startTime, endTime);
   dcl show: FashionShow := new FashionShow("theme", space, startTimeShow, endTimeShow, 10, 10,
       100);

  space.addStage(stage);
  show.addEvent(event1);
  show.cancelEvent("Keynote");
  -- show.cancelEvent("ClosingKeynote"); --> BREAKS (can't cancel an event that is not happening
      in this show)

  expecting(show.events, {});
 );

  -- tests requirement R17

 private testBuyTicket: () ==> ()
 testBuyTicket() == (
   dcl space: Space := new Space("name", "address");
   dcl show: FashionShow := new FashionShow("theme", space, mk_Util`Date(1,1,1,1,1), mk_Util`Date
       (1,1,1,1,2), 10, 1, 10);
   dcl person1: Person := new Person("name1", 1);
    dcl person2: Person := new Person("name2", 2);

    person1.buyTicket(show, <Normal>);
   -- show.buyTicket(person2, <Normal>); --> BREAKS (fashion show with maximum of 1 Normal ticket
       , and there is no Normal tickets left)

  expecting(card show.ticketsSold(<Normal>), 1);
  expecting(card person1.tickets, 1);
  expecting(show.ticketsSold(<Normal>), person1.tickets);
 );


 private testEquals: () ==> ()
 testEquals() == (
   dcl space1: Space := new Space("name", "address");
   dcl space2: Space := new Space("name", "address");
   dcl show1: FashionShow := new FashionShow("theme", space1, mk_Util`Date(1,1,1,1,1), mk_Util`
       Date(1,1,1,1,2), 10, 10, 1);
   dcl show2: FashionShow := new FashionShow("differentTheme", space2, mk_Util`Date(1,1,1,1,1),
       mk_Util`Date(1,1,1,1,2), 10, 10, 1);

   expecting(FashionShow`equals(show1, show1), true);
   expecting(FashionShow`equals(show1, show2), false);
 );

  -- tests Requirement R14

 private testGetAttendingDesigners: () ==> ()
 testGetAttendingDesigners() == (
   dcl space: Space := new Space("name", "address");
   dcl show: FashionShow := new FashionShow("theme", space, mk_Util`Date(1,1,1,1,1), mk_Util`Date
       (1,1,1,1,2), 10, 10, 1);
   dcl person1: Person := new Person("name1", 1);
    dcl person2: Person := new Person("name2", 2);
   dcl cloth: Cloth := new Cloth(person1, "Coat", <Winter>);

   person1.buyTicket(show, <Normal>);
   person2.buyTicket(show, <Normal>);

   person1.addCloth(cloth);

  expecting(show.getAttendingDesigners(), {person1});
```

```
);

 -- tests Requirement R14

private testGetAttendingDesignersToEvent: () ==> ()
testGetAttendingDesignersToEvent() == (
  dcl space: Space := new Space("name", "address");
  dcl startTimeShow: Util`Date := mk_Util`Date(1997,02,13,20,55);
  dcl endTimeShow: Util`Date := mk_Util`Date(1997,02,15,22,00);
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
  dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
  dcl stage: Stage := new Stage("B001", 200);
  dcl event: Event := new Event("Keynote", stage, startTime, endTime);
  dcl show: FashionShow := new FashionShow("theme", space, startTimeShow, endTimeShow, 10, 10,
      1);
  dcl person1: Person := new Person("name1", 1);
   dcl person2: Person := new Person("name2", 2);
   dcl person3: Person := new Person("name3", 2);
  dcl cloth1: Cloth := new Cloth(person1, "Coat", <Winter>);
  dcl cloth2: Cloth := new Cloth(person2, "Coat", <Winter>);

 space.addStage(stage);

  person1.addCloth(cloth1);
  person2.addCloth(cloth2);

  person1.buyTicket(show, <Normal>);
  person2.buyTicket(show, <Normal>);
  person3.buyTicket(show, <Normal>);

 show.addEvent(event);

 person1.attend(event);
 person3.attend(event);

 expecting(show.getAttendingDesignersToEvent(event), {person1});
);

 -- tests Requirement R14

private testGetParticipatingDesigners: () ==> ()
testGetParticipatingDesigners() == (
  dcl space: Space := new Space("name", "address");
  dcl startTimeShow: Util`Date := mk_Util`Date(1997,02,13,20,55);
  dcl endTimeShow: Util`Date := mk_Util`Date(1997,02,15,22,00);
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
  dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
  dcl stage: Stage := new Stage("B001", 200);
  dcl show: FashionShow := new FashionShow("theme", space, startTimeShow, endTimeShow, 10, 10,
      1);

    dcl runwayEvent: Runway := new Runway("WinterRunway", stage, startTime, endTime);

  dcl person1: Person := new Person("name1", 1);
   dcl person2: Person := new Person("name2", 2);
   dcl person3: Person := new Person("name3", 2);
  dcl cloth1: Cloth := new Cloth(person1, "Coat", <Winter>);
  dcl cloth2: Cloth := new Cloth(person2, "Coat", <Winter>);

 space.addStage(stage);

  person1.addCloth(cloth1);
  person2.addCloth(cloth2);

  person1.buyTicket(show, <Normal>);
  person2.buyTicket(show, <Normal>);
  person3.buyTicket(show, <Normal>);

  show.addEvent(runwayEvent);

 runwayEvent.addWorker(person1);
 runwayEvent.addWorker(person2);
 runwayEvent.addWorker(person3);

 runwayEvent.addToSequence(cloth1, person3);
 runwayEvent.addToSequence(cloth2, person3);

 expecting(show.getParticipatingDesigners(), {person1, person2});
);


private testGetWorkers: () ==> ()
testGetWorkers() == (
  dcl space: Space := new Space("name", "address");
  dcl startTimeShow: Util`Date := mk_Util`Date(1997,02,13,20,55);
  dcl endTimeShow: Util`Date := mk_Util`Date(1997,02,15,22,00);
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
  dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
  dcl stage: Stage := new Stage("B001", 200);
  dcl event: Event := new Event("Keynote", stage, startTime, endTime);
  dcl show: FashionShow := new FashionShow("theme", space, startTimeShow, endTimeShow, 10, 10,
      1);
  dcl person1: Person := new Person("name1", 1);
   dcl person2: Person := new Person("name2", 2);

 space.addStage(stage);

 show.addEvent(event);

 event.addWorker(person1);
 event.addWorker(person2);

 expecting(show.getWorkers(), {person1, person2});
);

 -- tests requirement R21

private testInviteGuest: () ==> ()
testInviteGuest() == (
  dcl space: Space := new Space("name", "address");
  dcl startTimeShow: Util`Date := mk_Util`Date(1997,02,13,20,55);
  dcl endTimeShow: Util`Date := mk_Util`Date(1997,02,15,22,00);
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
  dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
  dcl stage: Stage := new Stage("B001", 200);
   dcl presentation: Presentation := new Presentation("Keynote", stage, startTime, endTime, "
       Opening");
  dcl show: FashionShow := new FashionShow("theme", space, startTimeShow, endTimeShow, 10, 10,
      10);
  dcl person1: Person := new Person("name1", 1);
   dcl person2: Person := new Person("name2", 2);

 space.addStage(stage);

 show.addEvent(presentation);

 show.inviteGuest(person1, presentation);
 show.inviteGuest(person2, nil);

   expecting(card show.ticketsSold(<Guest>), 2);
```

```
    expecting(card person1.tickets, 1);
    expecting(card person2.tickets, 1);
    expecting(presentation.speakers, {person1});
 );

 -- tests requirement R17

 private testSetNEntries: () ==> ()
 testSetNEntries() == (
  dcl space: Space := new Space("name", "address");
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
  dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);

  dcl show: FashionShow := new FashionShow("theme", space, startTime, endTime, 10, 10, 10);

  show.setNEntries(<VIP>, 20);
  show.setNEntries(<Normal>, 30);
  show.setNEntries(<Guest>, 40);

 expecting(show.nEntries(<VIP>), 20);
 expecting(show.nEntries(<Normal>), 30);
 expecting(show.nEntries(<Guest>), 40);
 );

 -- tests requirement R20

 private testSetTheme: () ==> ()
 testSetTheme() == (
  dcl space: Space := new Space("name", "address");
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
  dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);

  dcl show: FashionShow := new FashionShow("theme", space, startTime, endTime, 10, 10, 10);

  show.setTheme("newTheme");

  expecting(show.theme, "newTheme");
 );


 private testSetSpace: () ==> ()
 testSetSpace() == (
  dcl space1: Space := new Space("name1", "address1");
  dcl space2: Space := new Space("name2", "address1");
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
  dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);

  dcl show: FashionShow := new FashionShow("theme", space1, startTime, endTime, 10, 10, 10);

  show.setSpace(space2);

  expecting(show.space, space2);
 );

 -- tests requirement R20

 private testSetTime: () ==> ()
 testSetTime() == (
  dcl space: Space := new Space("name", "address");
  dcl startTimeShow: Util`Date := mk_Util`Date(1997,02,13,20,55);
  dcl endTimeShow: Util`Date := mk_Util`Date(1997,02,15,22,00);
  dcl startTimeEvent1: Util`Date := mk_Util`Date(1997,02,13,21,00);
  dcl endTimeEvent1: Util`Date := mk_Util`Date(1997,02,13,21,55);
  dcl startTimeEvent2: Util`Date := mk_Util`Date(1997,02,14,21,00);
  dcl endTimeEvent2: Util`Date := mk_Util`Date(1997,02,14,21,55);
```

```
    dcl startTimeEvent3: Util`Date := mk_Util`Date(1997,02,15,21,00);
    dcl endTimeEvent3: Util`Date := mk_Util`Date(1997,02,15,21,55);
    dcl stage: Stage := new Stage("B001", 200);

    dcl event1: Event := new Event("Keynote", stage, startTimeEvent1, endTimeEvent1);
    dcl event2: Event := new Event("Keynote", stage, startTimeEvent2, endTimeEvent2);
    dcl event3: Event := new Event("Keynote", stage, startTimeEvent3, endTimeEvent3);
    dcl show: FashionShow := new FashionShow("theme", space, startTimeShow, endTimeShow, 10, 10,
        100);

    dcl newStartTimeShow: Util`Date := mk_Util`Date(1997,02,12,20,55);
    dcl newEndTimeShow: Util`Date := mk_Util`Date(1997,02,16,22,00);

    dcl impossibleStartTimeShow: Util`Date := mk_Util`Date(1997,02,14,20,55);
    dcl impossibleEndTimeShow: Util`Date := mk_Util`Date(1997,02,14,22,00);

   space.addStage(stage);
   show.addEvent(event1);
   show.addEvent(event2);
   show.addEvent(event3);

   show.setTime(newStartTimeShow, newEndTimeShow);
   -- show.setTime(newEndTimeShow, newStartTimeShow); --> BREAKS (start time must come before end
       time)
   -- show.setTime(impossibleStartTimeShow, impossibleEndTimeShow); --> BREAKS (all events must be
        contained within new start and end time)

   expecting(show.startTime, newStartTimeShow);
   expecting(show.endTime, newEndTimeShow);
   );


  public main:() ==> ()
  main() == (
   testConstructor();
   testEventsManager();
   testAddEvent(); -- R18
   testCancelEvent(); -- R18
   testBuyTicket(); -- R17
   testEquals();
   testGetAttendingDesigners(); -- R14
   testGetAttendingDesignersToEvent(); -- R14
   testGetParticipatingDesigners(); -- R14
   testGetWorkers();
   testInviteGuest(); -- R21
   testSetNEntries(); -- R17
   testSetTheme(); -- R20
   testSetSpace();
   testSetTime(); -- R20
  );
 end TestFashionShow
```

Function or operation	Line	Coverage	Calls
assert	8	100.0%	1360
main	409	100.0%	27
testAddEvent	111	100.0%	27
testBuyTicket	158	100.0%	27
testCancelEvent	136	100.0%	27
testConstructor	12	100.0%	85

testEquals	173	100.0%	27
testEventsManager	35	100.0%	85
testGetAttendingDesigners	185	100.0%	27
testGetAttendingDesignersToEvent	202	100.0%	27
testGetParticipatingDesigners	236	100.0%	27
testGetWorkers	275	100.0%	20
testInviteGuest	299	100.0%	27
testSetNEntries	326	100.0%	27
testSetSpace	357	100.0%	20
testSetTheme	344	100.0%	27
testSetTime	372	100.0%	28
TestFashionShow.vdmpp		100.0%	1895

18 TestNotification

```
class TestNotification is subclass of TestClass
operations

  private testEquals: () ==> ()
  testEquals() == (
  dcl date: Util`Date := mk_Util`Date(1997,02,14,20,55);
   dcl n1: Notification := new Notification("title1", "description1", date, <Workers>);
   dcl n2: Notification := new Notification("title1", "description1", date, <Workers>);
   dcl n3: Notification := new Notification("title2", "description1", date, <Workers>);
   dcl n4: Notification := new Notification("title1", "description2", date, <Workers>);
   dcl n5: Notification := new Notification("title1", "description1", date, <Attendees>);

   expecting(Notification`equals(n1, n2), true);
   expecting(Notification`equals(n1, n3), false);
   expecting(Notification`equals(n1, n4), false);
   expecting(Notification`equals(n1, n5), false);
  );


  private testSetTitle: () ==> ()
  testSetTitle() == (
  dcl date: Util`Date := mk_Util`Date(1997,02,14,20,55);
   dcl n: Notification := new Notification("title", "description", date, <Workers>);

   n.setTitle("different title");
   expecting(n.title, "different title");
  );


  private testSetDescription: () ==> ()
  testSetDescription() == (
  dcl date: Util`Date := mk_Util`Date(1997,02,14,20,55);
   dcl n: Notification := new Notification("title", "description", date, <Workers>);

   n.setDescription("different description");
   expecting(n.description, "different description");
  );


  private testSetDate: () ==> ()
  testSetDate() == (
  dcl date: Util`Date := mk_Util`Date(1997,02,14,20,55);
  dcl newDate: Util`Date := mk_Util`Date(1997,02,14,21,55);
   dcl n: Notification := new Notification("title", "description", date, <Workers>);

   n.setDate(newDate);
   expecting(n.date, newDate);
  );


  public static main: () ==> ()
 main() == (
   dcl t: TestNotification := new TestNotification();
   t.testEquals();
   t.testSetTitle();
   t.testSetDescription();
   t.testSetDate();
  );

end TestNotification
```

Function or operation	Line	Coverage	Calls
main	46	100.0%	119
testEquals	3	100.0%	119
testSetDate	36	100.0%	119
testSetDescription	27	100.0%	119
testSetTitle	18	100.0%	119
TestNotification.vdmpp		100.0%	595

19 TestPerson

```
class TestPerson is subclass of TestClass
operations

  private testGetJobNotifications: () ==> ()
  testGetJobNotifications() == (
   dcl space: Space := new Space("name", "address");
   dcl person: Person := new Person("name", 1);
   dcl startTimeShow1: Util`Date := mk_Util`Date(1997,02,13,20,55);
   dcl endTimeShow1: Util`Date := mk_Util`Date(1997,02,15,22,00);
   dcl startTime1: Util`Date := mk_Util`Date(1997,02,14,21,00);
   dcl endTime1: Util`Date := mk_Util`Date(1997,02,14,21,55);
   dcl startTimeShow2: Util`Date := mk_Util`Date(1998,02,13,20,55);
   dcl endTimeShow2: Util`Date := mk_Util`Date(1998,02,15,22,00);
   dcl startTime2: Util`Date := mk_Util`Date(1998,02,14,21,00);
   dcl endTime2: Util`Date := mk_Util`Date(1998,02,14,21,55);
   dcl stage: Stage := new Stage("B001", 200);

   dcl event1: Event := new Event("Keynote", stage, startTime1, endTime1);
   dcl event2: Event := new Event("Keynote", stage, startTime2, endTime2);
   dcl show1: FashionShow := new FashionShow("theme", space, startTimeShow1, endTimeShow1, 10,
       10, 100);
   dcl show2: FashionShow := new FashionShow("theme", space, startTimeShow2, endTimeShow2, 10,
       10, 100);

    dcl notificationTime1: Util`Date := mk_Util`Date(1997,02,14,21,00);
    dcl notification1: Notification := new Notification("title", "Keynote started",
        notificationTime1, <Workers>);
    dcl notification2: Notification := new Notification("title", "Keynote ended",
        notificationTime1, <Workers>);

    dcl notificationTime2: Util`Date := mk_Util`Date(1998,02,14,21,00);
    dcl notification3: Notification := new Notification("title", "Keynote started",
        notificationTime2, <Attendees>);
    dcl notification4: Notification := new Notification("title", "Keynote ended",
        notificationTime2, <Attendees>);

  space.addStage(stage);

  show1.addEvent(event1);
  event1.addWorker(person);
    event1.addNotification(notification1);
    event1.addNotification(notification2);

  show2.addEvent(event2);
  person.buyTicket(show2, <Normal>);
  person.attend(event2);
    event2.addNotification(notification3);
    event2.addNotification(notification4);

  expecting(person.getJobNotifications(), {notification1, notification2});
  expecting(person.getAttendeeNotifications(), {notification3, notification4});
  );


  private testGetAttendanceNotifications: () ==> ()
  testGetAttendanceNotifications() == (
   dcl space: Space := new Space("name", "address");
   dcl person: Person := new Person("name", 1);
   dcl startTimeShow1: Util`Date := mk_Util`Date(1997,02,13,20,55);
   dcl endTimeShow1: Util`Date := mk_Util`Date(1997,02,15,22,00);
   dcl startTime1: Util`Date := mk_Util`Date(1997,02,14,21,00);
   dcl endTime1: Util`Date := mk_Util`Date(1997,02,14,21,55);
   dcl startTimeShow2: Util`Date := mk_Util`Date(1998,02,13,20,55);
   dcl endTimeShow2: Util`Date := mk_Util`Date(1998,02,15,22,00);
   dcl startTime2: Util`Date := mk_Util`Date(1998,02,14,21,00);
   dcl endTime2: Util`Date := mk_Util`Date(1998,02,14,21,55);
   dcl stage: Stage := new Stage("B001", 200);

   dcl event1: Event := new Event("Keynote", stage, startTime1, endTime1);
   dcl event2: Event := new Event("Keynote", stage, startTime2, endTime2);
   dcl show1: FashionShow := new FashionShow("theme", space, startTimeShow1, endTimeShow1, 10,
       10, 100);
   dcl show2: FashionShow := new FashionShow("theme", space, startTimeShow2, endTimeShow2, 10,
       10, 100);

    dcl notificationTime1: Util`Date := mk_Util`Date(1997,02,14,21,00);
    dcl notification1: Notification := new Notification("title", "Keynote started",
        notificationTime1, <Attendees>);
    dcl notification2: Notification := new Notification("title", "Keynote ended",
        notificationTime1, <Attendees>);

    dcl notificationTime2: Util`Date := mk_Util`Date(1998,02,14,21,00);
    dcl notification3: Notification := new Notification("title", "Keynote started",
        notificationTime2, <Attendees>);
    dcl notification4: Notification := new Notification("title", "Keynote ended",
        notificationTime2, <Attendees>);

  space.addStage(stage);

  show1.addEvent(event1);
```

```
person.buyTicket(show1, <Normal>);
person.attend(event1);
  event1.addNotification(notification1);
  event1.addNotification(notification2);

show2.addEvent(event2);
person.buyTicket(show2,<Normal>);
person.attend(event2);
  event2.addNotification(notification3);
  event2.addNotification(notification4);

expecting(person.getAttendanceNotifications(show1), {notification1, notification2});
expecting(person.getAttendanceNotifications(show2), {notification3, notification4});
);


private testAddTicket: () ==> ()
testAddTicket() == (
 dcl space: Space := new Space("name", "address");
 dcl show: FashionShow := new FashionShow("theme", space, mk_Util`Date(1,1,1,1,1), mk_Util`Date
     (1,1,1,1,2), 10, 10, 10);
 dcl person: Person := new Person("name", 1);
 dcl ticket: Ticket := new Ticket(<Normal>, person, show);

 person.addTicket(ticket);
 expectingIn(ticket, person.tickets);
);

-- tests requirement R5

private testAddJob: () ==> ()
testAddJob() == (
 dcl space: Space := new Space("name", "address");
 dcl show: FashionShow := new FashionShow("theme", space, mk_Util`Date(1,1,1,1,1), mk_Util`Date
     (1,1,1,1,2), 10, 10, 10);
 dcl person: Person := new Person("name", 1);
 dcl startTime: Util`Date := mk_Util`Date(1997,02,14,20,55);
 dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
 dcl stage: Stage := new Stage("B001", 200);
 dcl stage2: Stage := new Stage("B002", 200);
 dcl event: Event := new Event("Keynote", stage, startTime, endTime);
 dcl event2: Event := new Event("Keynote", stage2, startTime, endTime);

 person.addJob(event);
 -- person.addJob(event2); --> BREAKS (can't add two jobs at the same time)
 expectingIn(event, person.jobs);
);


private testRemoveJob: () ==> ()
testRemoveJob() == (
 dcl space: Space := new Space("name", "address");
 dcl show: FashionShow := new FashionShow("theme", space, mk_Util`Date(1,1,1,1,1), mk_Util`Date
     (1,1,1,1,2), 10, 10, 10);
 dcl person: Person := new Person("name", 1);
 dcl startTime: Util`Date := mk_Util`Date(1997,02,14,20,55);
 dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
 dcl stage: Stage := new Stage("B001", 200);
 dcl event: Event := new Event("Keynote", stage, startTime, endTime);

 person.addJob(event);
 person.removeJob(event);
 expectingNotIn(event, person.jobs);
);
```

```
private testAttend: () ==> ()
testAttend() == (
 dcl space: Space := new Space("name", "address");
 dcl person: Person := new Person("name", 1);
 dcl startTimeShow1: Util`Date := mk_Util`Date(1997,02,13,20,55);
 dcl endTimeShow1: Util`Date := mk_Util`Date(1997,02,15,22,00);
 dcl startTime1: Util`Date := mk_Util`Date(1997,02,14,21,00);
 dcl endTime1: Util`Date := mk_Util`Date(1997,02,14,21,55);
 dcl startTimeShow2: Util`Date := mk_Util`Date(1998,02,13,20,55);
 dcl endTimeShow2: Util`Date := mk_Util`Date(1998,02,15,22,00);
 dcl startTime2: Util`Date := mk_Util`Date(1998,02,14,21,00);
 dcl endTime2: Util`Date := mk_Util`Date(1998,02,14,21,55);
 dcl stage: Stage := new Stage("B001", 200);

 dcl event1: Event := new Event("Keynote", stage, startTime1, endTime1);
 dcl event2: Event := new Event("Keynote", stage, startTime2, endTime2);
 dcl show1: FashionShow := new FashionShow("theme", space, startTimeShow1, endTimeShow1, 10,
     10, 100);
 dcl show2: FashionShow := new FashionShow("theme", space, startTimeShow2, endTimeShow2, 10,
     10, 100);

space.addStage(stage);

show1.addEvent(event1);
person.buyTicket(show1,<Normal>);
person.attend(event1);

show2.addEvent(event2);
person.addTicket(new Ticket(<Guest>, person, show2));
person.attend(event2);

expecting({event1}, person.attendance(show1));
expecting({event2}, person.attendance(show2));
);



private testCancelAttendment: () ==> ()
testCancelAttendment() == (
 dcl space: Space := new Space("name", "address");
 dcl person: Person := new Person("name", 1);
 dcl startTimeShow: Util`Date := mk_Util`Date(1997,02,13,20,55);
 dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
 dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
 dcl endTimeShow: Util`Date := mk_Util`Date(1997,02,15,22,00);
 dcl stage: Stage := new Stage("B001", 200);
 dcl event: Event := new Event("Keynote", stage, startTime, endTime);
 dcl show: FashionShow := new FashionShow("theme", space, startTimeShow, endTimeShow, 10, 10,
     100);

space.addStage(stage);
show.addEvent(event);
person.buyTicket(show, <Normal>);
person.attend(event);
person.cancelAttendment(event);

expectingNotIn(event, person.attendance(show));
);


private testAddCloth: () ==> ()
testAddCloth() == (
 dcl person: Person := new Person("Designer", 1);
 dcl cloth: Cloth := new Cloth(person, "Coat", <Winter>);
 person.addCloth(cloth);

 expectingIn(cloth, person.designed);
);


private testIsDesigner: () ==> ()
testIsDesigner() == (
 dcl person: Person := new Person("Designer", 1);
 dcl cloth: Cloth := new Cloth(person, "Coat", <Winter>);

 expecting(person.isDesigner(), false);
 person.designed := {cloth};
 expecting(person.isDesigner(), true);
);


private testIsParticipantDesigner: () ==> ()
testIsParticipantDesigner() == (
 dcl space: Space := new Space("name", "address");
 dcl startTimeShow: Util`Date := mk_Util`Date(1997,02,13,20,55);
 dcl endTimeShow: Util`Date := mk_Util`Date(1997,02,15,22,00);
 dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
 dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
 dcl stage: Stage := new Stage("B001", 200);
 dcl show: FashionShow := new FashionShow("theme", space, startTimeShow, endTimeShow, 10, 10,
     1);

   dcl runwayEvent: Runway := new Runway("WinterRunway", stage, startTime, endTime);

 dcl person1: Person := new Person("name1", 1);
  dcl person2: Person := new Person("name2", 2);
  dcl person3: Person := new Person("name3", 2);
 dcl cloth1: Cloth := new Cloth(person1, "Coat", <Winter>);
 dcl cloth2: Cloth := new Cloth(person2, "Coat", <Winter>);

space.addStage(stage);

 person1.addCloth(cloth1);
 person2.addCloth(cloth2);

 show.addEvent(runwayEvent);

runwayEvent.addWorker(person1);
runwayEvent.addWorker(person2);
runwayEvent.addWorker(person3);

runwayEvent.addToSequence(cloth1, person3);
runwayEvent.addToSequence(cloth2, person3);

 expecting(person1.isParticipantDesigner(), {runwayEvent});
);


private testCancelPresentation: () ==> ()
testCancelPresentation() == (
dcl space: Space := new Space("name", "address");
 dcl startTimeShow: Util`Date := mk_Util`Date(1997,02,13,20,55);
 dcl endTimeShow: Util`Date := mk_Util`Date(1997,02,15,22,00);
 dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
 dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
 dcl stage: Stage := new Stage("B001", 200);
 dcl show: FashionShow := new FashionShow("theme", space, startTimeShow, endTimeShow, 10, 10,
     1);
```

```
    dcl pres: Presentation := new Presentation("WinterRunway", stage, startTime, endTime,"
        subtheme");
       dcl person1: Person := new Person("name1", 1);
       dcl person2 : Person := new Person("n2",2);
    space.addStage(stage);
   show.addEvent(pres);
  pres.addSpeaker(person1);
  pres.addSpeaker(person2);

  expecting(pres in set person1.speaks,true);
  expecting(pres in set person2.speaks,true);

  person1.cancelPresentation(pres);
  person2.cancelPresentation(pres);
  expecting(pres not in set person1.speaks,true);
  expecting(pres not in set person2.speaks,true);
 );


 private testIsWorking: () ==> ()
 testIsWorking() == (
  dcl startTime1: Util`Date := mk_Util`Date(1997,02,14,21,00);
  dcl endTime1: Util`Date := mk_Util`Date(1997,02,14,21,55);
  dcl startTime2: Util`Date := mk_Util`Date(1997,02,13,20,55);
  dcl endTime2: Util`Date := mk_Util`Date(1997,02,15,22,00);
  dcl startTime3: Util`Date := mk_Util`Date(2009,02,13,20,55);
  dcl endTime3: Util`Date := mk_Util`Date(2009,02,15,22,00);
  dcl stage: Stage := new Stage("B001", 200);
  dcl event: Event := new Event("Keynote", stage, startTime1, endTime1);
  dcl person: Person := new Person("name", 1);

  expecting(person.isWorking(startTime2, endTime2, {event},{}), true);
  expecting(person.isWorking(startTime3, endTime3, {event},{}), false);
 );



 public static main: () ==> ()
main() == (
  dcl t: TestPerson := new TestPerson();
  t.testGetJobNotifications();
  t.testGetAttendanceNotifications();
  t.testAddTicket();
  t.testAddJob(); -- R5
  t.testRemoveJob();
  t.testAttend();
  t.testCancelAttendment();
  t.testAddCloth();
  t.testIsDesigner();
  t.testIsParticipantDesigner();
  t.testIsWorking();
  t.testCancelPresentation();
 );

end TestPerson
```

Function or operation	Line	Coverage	Calls
main	290	100.0%	2
testAddCloth	192	100.0%	54

testAddJob	104	100.0%	54
testAddTicket	92	100.0%	152
testAttend	137	100.0%	40
testCancelAttendment	171	100.0%	54
testCancelPresentation	246	100.0%	2
testGetAttendanceNotifications	47	100.0%	152
testGetJobNotifications	3	100.0%	158
testIsDesigner	201	100.0%	54
testIsParticipantDesigner	211	100.0%	18
testIsWorking	273	100.0%	2
testRemoveJob	121	100.0%	54
TestPerson.vdmpp		100.0%	796

20 TestPresentation

```
class TestPresentation is subclass of TestClass
operations
 -- tests requirement R23

 private testAddSpeaker: () ==> ()
  testAddSpeaker() == (
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,20,55);
   dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
   dcl stage: Stage := new Stage("B001", 200);
   dcl presentation: Presentation := new Presentation("Keynote", stage, startTime, endTime, "
       Opening");
   dcl speaker: Person := new Person("Person", 1);

   expectingNotIn(speaker, presentation.workers);
   expectingNotIn(speaker, presentation.speakers);

   presentation.addSpeaker(speaker);

   expectingIn(speaker, presentation.workers);
   expectingIn(speaker, presentation.speakers);
  );

  -- tests requirement R20

  private testUpdateSubTheme: () ==> ()
  testUpdateSubTheme() == (
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,20,55);
   dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
   dcl stage: Stage := new Stage("B001", 200);
   dcl presentation: Presentation := new Presentation("Keynote", stage, startTime, endTime, "
       Opening");

   presentation.updateSubTheme("Closing");
   expecting(presentation.subTheme, "Closing");
  );


  private testDismissWorker: () ==> ()
  testDismissWorker() == (
   dcl startTime: Util`Date := mk_Util`Date(1997,02,14,20,55);
   dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
   dcl stage: Stage := new Stage("B001", 200);
```

```
    dcl presentation: Presentation := new Presentation("Keynote", stage, startTime, endTime,
        "Opening");
    dcl p1 : Person := new Person("n",1);
    dcl p2 : Person := new Person("n1",2);
    presentation.addWorker(p1);
    presentation.addWorker(p2);

    presentation.addSpeaker(p1);
    presentation.addWorker(p2);

    presentation.dismissWorker(p1);
    presentation.dismissWorker(p2);

    expecting(p1 not in set presentation.workers,true);
    expecting(p2 not in set presentation.workers and p2 not in set presentation.speakers,true);
  );


  public static main: () ==> ()
 main() == (
    dcl t: TestPresentation := new TestPresentation();
    t.testAddSpeaker(); -- R23
    t.testUpdateSubTheme(); -- R20
  );
end TestPresentation
```

Function or operation	Line	Coverage	Calls
main	54	100.0%	2
testAddSpeaker	4	100.0%	32
testDismissWorker	33	0.0%	0
testUpdateSubTheme	22	100.0%	32
TestPresentation.vdmpp		57.0%	66

21 TestRunway

```
class TestRunway is subclass of TestClass
operations
  -- tests requirement R22

 private testAddToSequence: () ==> ()
  testAddToSequence() == (
    dcl space: Space := new Space("name", "address");
    dcl startTimeShow: Util`Date := mk_Util`Date(1997,02,13,20,55);
    dcl endTimeShow: Util`Date := mk_Util`Date(1997,02,15,22,00);
    dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
    dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
    dcl stage: Stage := new Stage("B001", 200);
    dcl show: FashionShow := new FashionShow("theme", space, startTimeShow, endTimeShow, 10, 10,
        1);

     dcl runwayEvent: Runway := new Runway("WinterRunway", stage, startTime, endTime);

    dcl person1: Person := new Person("name1", 1);
     dcl person2: Person := new Person("name2", 2);
    dcl cloth: Cloth := new Cloth(person1, "Coat", <Winter>);
   space.addStage(stage);

    person1.addCloth(cloth);

    person1.buyTicket(show, <Normal>);
    person2.buyTicket(show, <Normal>);

    show.addEvent(runwayEvent);

   runwayEvent.addWorker(person1);
   runwayEvent.addWorker(person2);

   runwayEvent.addToSequence(cloth, person2);

    expecting(len runwayEvent.models, 1);
    expecting(runwayEvent.models(1).cloth, cloth);
    expecting(runwayEvent.models(1).worker, person2);
   );

   -- tests requirement R22

   private testRemoveFromSequence: () ==> ()
   testRemoveFromSequence() == (
    dcl space: Space := new Space("name", "address");
    dcl startTimeShow: Util`Date := mk_Util`Date(1997,02,13,20,55);
    dcl endTimeShow: Util`Date := mk_Util`Date(1997,02,15,22,00);
    dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
    dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
    dcl stage: Stage := new Stage("B001", 200);
    dcl show: FashionShow := new FashionShow("theme", space, startTimeShow, endTimeShow, 10, 10,
        1);

     dcl runwayEvent: Runway := new Runway("WinterRunway", stage, startTime, endTime);

    dcl person1: Person := new Person("name1", 1);
     dcl person2: Person := new Person("name2", 2);
     dcl person3: Person := new Person("name3", 3);
    dcl cloth1: Cloth := new Cloth(person1, "Coat", <Winter>);
    dcl cloth2: Cloth := new Cloth(person1, "Coat", <Winter>);

   space.addStage(stage);

    person1.addCloth(cloth1);
    person1.addCloth(cloth2);

    person1.buyTicket(show, <Normal>);
    person2.buyTicket(show, <Normal>);
    person3.buyTicket(show, <Normal>);

    show.addEvent(runwayEvent);

   runwayEvent.addWorker(person1);
   runwayEvent.addWorker(person2);
   runwayEvent.addWorker(person3);

   runwayEvent.addToSequence(cloth1, person2);
   runwayEvent.addToSequence(cloth2, person3);

   runwayEvent.removeFromSequence(mk_Runway`Model(person2, cloth1));

    expecting(len runwayEvent.models, 1);
   );

   -- tests requirement R19
```

```
private testDismissWorker: () ==> ()
 testDismissWorker() == (
  dcl space: Space := new Space("name", "address");
  dcl startTimeShow: Util`Date := mk_Util`Date(1997,02,13,20,55);
  dcl endTimeShow: Util`Date := mk_Util`Date(1997,02,15,22,00);
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
  dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
  dcl stage: Stage := new Stage("B001", 200);
  dcl show: FashionShow := new FashionShow("theme", space, startTimeShow, endTimeShow, 10, 10,
      1);

   dcl runwayEvent: Runway := new Runway("WinterRunway", stage, startTime, endTime);

  dcl person1: Person := new Person("name1", 1);
   dcl person2: Person := new Person("name2", 2);
  dcl cloth: Cloth := new Cloth(person1, "Coat", <Winter>);

 space.addStage(stage);

  person1.addCloth(cloth);

  person1.buyTicket(show, <Normal>);
  person2.buyTicket(show, <Normal>);

  show.addEvent(runwayEvent);

 runwayEvent.addWorker(person1);
 runwayEvent.addWorker(person2);

 runwayEvent.addToSequence(cloth, person2);

 runwayEvent.dismissWorker(person2);

  expecting(len runwayEvent.models, 0);
  expectingNotIn(person2, runwayEvent.workers);
 );



private testRemoveWorkerModel: () ==> ()
 testRemoveWorkerModel() == (
  dcl space: Space := new Space("name", "address");
  dcl startTimeShow: Util`Date := mk_Util`Date(1997,02,13,20,55);
  dcl endTimeShow: Util`Date := mk_Util`Date(1997,02,15,22,00);
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
  dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
  dcl stage: Stage := new Stage("B001", 200);
  dcl show: FashionShow := new FashionShow("theme", space, startTimeShow, endTimeShow, 10, 10,
      1);

   dcl runwayEvent: Runway := new Runway("WinterRunway", stage, startTime, endTime);

  dcl person1: Person := new Person("name1", 1);
   dcl person2: Person := new Person("name2", 2);
   dcl person3: Person := new Person("name3", 3);
  dcl cloth1: Cloth := new Cloth(person1, "Coat", <Winter>);
  dcl cloth2: Cloth := new Cloth(person1, "Coat", <Winter>);

 space.addStage(stage);

  person1.addCloth(cloth1);
  person1.addCloth(cloth2);

  person1.buyTicket(show, <Normal>);
  person2.buyTicket(show, <Normal>);
  person3.buyTicket(show, <Normal>);

  show.addEvent(runwayEvent);

 runwayEvent.addWorker(person1);
 runwayEvent.addWorker(person2);
 runwayEvent.addWorker(person3);

 runwayEvent.addToSequence(cloth1, person2);
 runwayEvent.addToSequence(cloth2, person3);

 runwayEvent.dismissWorker(person2);
 -- runwayEvent.dismissWorker(person2); --> BREAKS (can't dismiss a person that is not working
     in this Event)

  expecting(len runwayEvent.models, 1);
 );


 private testGetClothes: () ==> ()
 testGetClothes() == (

  dcl space: Space := new Space("name", "address");
  dcl startTimeShow: Util`Date := mk_Util`Date(1997,02,13,20,55);
  dcl endTimeShow: Util`Date := mk_Util`Date(1997,02,15,22,00);
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
  dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
  dcl stage: Stage := new Stage("B001", 200);
  dcl show: FashionShow := new FashionShow("theme", space, startTimeShow, endTimeShow, 10, 10,
      1);

   dcl runwayEvent: Runway := new Runway("WinterRunway", stage, startTime, endTime);

  dcl person1: Person := new Person("name1", 1);
   dcl person2: Person := new Person("name2", 2);
   dcl person3: Person := new Person("name3", 3);
  dcl cloth1: Cloth := new Cloth(person1, "Coat", <Winter>);
  dcl cloth2: Cloth := new Cloth(person1, "Coat", <Winter>);

 space.addStage(stage);

  person1.addCloth(cloth1);
  person1.addCloth(cloth2);

  person1.buyTicket(show, <Normal>);
  person2.buyTicket(show, <Normal>);
  person3.buyTicket(show, <Normal>);

  show.addEvent(runwayEvent);

 runwayEvent.addWorker(person1);
 runwayEvent.addWorker(person2);
 runwayEvent.addWorker(person3);

 runwayEvent.addToSequence(cloth1, person2);
 runwayEvent.addToSequence(cloth2, person3);

  expecting(runwayEvent.getClothes(), {cloth1, cloth2});
 );



 private testGetDesigners: () ==> ()
 testGetDesigners() == (
  dcl space: Space := new Space("name", "address");
```

```
  dcl startTimeShow: Util`Date := mk_Util`Date(1997,02,13,20,55);
  dcl endTimeShow: Util`Date := mk_Util`Date(1997,02,15,22,00);
  dcl startTime: Util`Date := mk_Util`Date(1997,02,14,21,00);
  dcl endTime: Util`Date := mk_Util`Date(1997,02,14,21,55);
  dcl stage: Stage := new Stage("B001", 200);
  dcl show: FashionShow := new FashionShow("theme", space, startTimeShow, endTimeShow, 10, 10,
      1);

   dcl runwayEvent: Runway := new Runway("WinterRunway", stage, startTime, endTime);

  dcl person1: Person := new Person("name1", 1);
   dcl person2: Person := new Person("name2", 2);
   dcl person3: Person := new Person("name3", 3);
  dcl cloth1: Cloth := new Cloth(person1, "Coat", <Winter>);
  dcl cloth2: Cloth := new Cloth(person1, "Coat", <Winter>);

 space.addStage(stage);

  person1.addCloth(cloth1);
  person1.addCloth(cloth2);

  person1.buyTicket(show, <Normal>);
  person2.buyTicket(show, <Normal>);
  person3.buyTicket(show, <Normal>);

  show.addEvent(runwayEvent);

 runwayEvent.addWorker(person1);
 runwayEvent.addWorker(person2);
 runwayEvent.addWorker(person3);

 runwayEvent.addToSequence(cloth1, person2);
 runwayEvent.addToSequence(cloth2, person3);

  expecting(runwayEvent.getDesigners(), {person1});
 );


private testEqualModels: () ==> ()
 testEqualModels() == (
  dcl person1: Person := new Person("name1", 1);
   dcl person2: Person := new Person("name2", 2);
  dcl cloth1: Cloth := new Cloth(person1, "Coat", <Winter>);
  dcl cloth2: Cloth := new Cloth(person1, "Coat", <Winter>);

  dcl model1: Runway`Model := mk_Runway`Model(person1, cloth1);
  dcl model2: Runway`Model := mk_Runway`Model(person1, cloth2);
  dcl model3: Runway`Model := mk_Runway`Model(person2, cloth2);
  dcl model4: Runway`Model := mk_Runway`Model(person1, cloth1);

  expecting(Runway`equalModels(model1, model1), true);
  expecting(Runway`equalModels(model1, model4), true);
  expecting(Runway`equalModels(model1, model2), false);
  expecting(Runway`equalModels(model1, model3), false);
  expecting(Runway`equalModels(model2, model3), false);
 );


 public static main: () ==> ()
main() == (
  dcl t: TestRunway := new TestRunway();
  t.testAddToSequence(); -- R22
  t.testRemoveFromSequence(); -- R22
  t.testDismissWorker(); -- R19
  t.testRemoveWorkerModel();
```

```
    t.testGetClothes();
    t.testGetDesigners();
    t.testEqualModels();
  );
end TestRunway
```

Function or operation	Line	Coverage	Calls
main	260	100.0%	56
testAddToSequence	4	100.0%	32
testDismissWorker	82	100.0%	32
testEqualModels	241	100.0%	32
testGetClothes	161	100.0%	32
testGetDesigners	202	100.0%	32
testRemoveFromSequence	40	100.0%	32
testRemoveWorkerModel	119	100.0%	32
TestRunway.vdmpp		100.0%	280

22 TestShowsManager

```
class TestShowsManager is subclass of TestClass

instance variables
public manager: ShowsManager := new ShowsManager();

operations

 -- tests requirement R15

 public testAddShows: () ==> ()
 testAddShows() == (
  dcl s1 : FashionShow, s2 : FashionShow, s3 : FashionShow, s4: FashionShow,
    theme : seq of char := "theme",
    d1: Util`Date, d2: Util`Date, d3: Util`Date, d4: Util`Date;

  d1 := mk_Util`Date(2017,02,24,10,30);
  d2 := mk_Util`Date(2017,03,1,18,00);
  d3 := mk_Util`Date(2017,03,5,18,00);
  d4 := mk_Util`Date(2017,03,10,20,00);

  s1 := new FashionShow(theme, new Space("name1","addr1"), d1, d3, 50, 10, 10);
  s2 := new FashionShow(theme, new Space("name1","addr1"), d2, d3, 50, 10, 10);  --time/address
      overlap (with s1) -> BREAK
  s3 := new FashionShow(theme, new Space("name2","addr2"), d1, d2, 50, 10, 10);  --same show
      theme with time overlaps (with s1) ->BREAK
  s4 := new FashionShow(theme, new Space("name1","addr1"), d3, d4, 50, 10, 10);  --address
      overlap (with s1)

  manager.addShow(s1);
  --  manager.addShow(s1); --Duplicated -> BREAKS
  -- manager.addShow(s2); --Time and address overlap -->BREAK
  --  manager.addShow(s3); --Same Theme with time overlaps -->BREAK
  manager.addShow(s4);

  return
 );
```

```
-- tests requirement R16

public testRemoveShows: () ==> ()
testRemoveShows() == (
 dcl s1: FashionShow := new FashionShow("t1",new Space("name1","addr1"),mk_Util`Date
      (2017,02,20,10,00), mk_Util`Date(2017,02,26,11,00),50,10,10),
   s2: FashionShow := new FashionShow("t2",new Space("name2","addr2"),mk_Util`Date
        (2017,02,20,10,00), mk_Util`Date(2017,02,26,11,00),50,10,10);

dcl stage: Stage := new Stage("B001", 200);
dcl event1: Event := new Event("Keynote", stage, mk_Util`Date(2017,02,20,10,00), mk_Util`Date
     (2017,02,26,11,00));
dcl person1: Person := new Person("name1", 1);

s1.space.addStage(stage);
s1.addEvent(event1);
person1.buyTicket(s1, <Normal>);

manager.addShow(s1);
manager.addShow(s2);

manager.removeShow(s1);
manager.removeShow(s2);
--manager.removeShow("t1"); --Already removed -> Doesn't Exits -> BREAK
--manager.removeShow("a"); --Doesn't exist -> BREAK
);


private testGetSpaces: () ==> ()
testGetSpaces() == (
 dcl s1 : FashionShow, s2 : FashionShow,
   theme : seq of char := "theme",
   d1: Util`Date, d2: Util`Date, d3: Util`Date,
   space1: Space, space2: Space;

 d1 := mk_Util`Date(2017,02,24,10,30);
 d2 := mk_Util`Date(2017,03,5,18,00);
 d3 := mk_Util`Date(2017,03,10,20,00);

 space1 := new Space("name1","addr1");
 space2 := new Space("name2","addr2");

 s1 := new FashionShow(theme, space1, d1, d2, 50, 10, 10);
 s2 := new FashionShow(theme, space2, d2, d3, 50, 10, 10);

 manager.addShow(s1);
 manager.addShow(s2);

 expecting(manager.getSpaces(), {space1, space2});
);


public static main: () ==> ()
main() == (
 dcl t : TestShowsManager := new TestShowsManager();
 t.testAddShows(); -- R15
 t.manager.reset();

 t.testRemoveShows(); -- R16
 t.manager.reset();

 t.testGetSpaces();
 t.manager.reset();
);
```

```
end TestShowsManager
```

Function or operation	Line	Coverage	Calls
main	80	100.0%	16
testAddShows	9	100.0%	154
testGetSpaces	57	100.0%	16
testRemoveShows	35	100.0%	16
TestShowsManager.vdmpp		100.0%	202

23 TestSpace

```
class TestSpace is subclass of TestClass
operations

  private testSetAddress: () ==> ()
  testSetAddress() == (
   dcl space: Space := new Space("name", "address");
   space.setAddress("different address");

   expecting(space.address, "different address");
  );


  private testSetName: () ==> ()
  testSetName() == (
   dcl space: Space := new Space("name", "address");
   space.setName("different name");

   expecting(space.name, "different name");
  );


  private testReset: () ==> ()
  testReset() == (
   dcl space: Space := new Space("name", "address");
   dcl show: FashionShow := new FashionShow("theme", space, mk_Util`Date(1,1,1,1,1), mk_Util`Date
       (1,1,1,1,2), 10, 10, 10);

  space.reset();
  expecting(space.reservations, {});
  );

   -- tests Requirement R27

   private testManageStages: () ==> ()
  testManageStages() == (
   dcl space: Space := new Space("name", "address");
   dcl stage1: Stage := new Stage("B001", 200);
   dcl stage2: Stage := new Stage("B002", 180);

   -- add stages
   space.addStage(stage1);
   space.addStage(stage2);
   -- space.addStage(stage2); --> BREAKS (can't add a stage that is already part of the space)
   expecting(space.stages, {stage1, stage2});
   -- edit stage name
   stage1.setName("Queijo 1");
   -- stage1.setName(""); --> BREAKS (name len = 0)
   expecting(stage1.name, "Queijo 1");

   -- edit stage seats
   stage2.setSeats(200);
   -- stage2.setSeats(0); --> BREAKS (seats is nat1)
   expecting(stage2.seats, 200);

   -- remove stages
   space.removeStage(stage1);
   -- space.removeStage(stage1); --> BREAKS (can't remove a stage that is not part of the space)
   expecting(space.stages, {stage2});
  );


  public static main: () ==> ()
 main() == (
   dcl t: TestSpace := new TestSpace();
   t.testSetAddress();
   t.testSetName();
   t.testReset();
   t.testManageStages(); -- R27
  );

end TestSpace
```

Function or operation	Line	Coverage	Calls
main	57	100.0%	34
testManageStages	29	100.0%	34
testReset	19	100.0%	98
testSetAddress	3	100.0%	124
testSetName	11	100.0%	124
TestSpace.vdmpp		100.0%	414

24 TestStage

```
class TestStage is subclass of TestClass
operations

  private testSetName: () ==> ()
  testSetName() == (
   dcl stage: Stage := new Stage("B001", 200);
   stage.setName("B002");

   expecting(stage.name, "B002");
  );


  private testSetSeats: () ==> ()
  testSetSeats() == (
   dcl stage: Stage := new Stage("B001", 200);
   stage.setSeats(300);
   expecting(stage.seats, 300);
  );


  private testEqual: () ==> ()
  testEqual() == (
   dcl stage1: Stage := new Stage("B001", 200);
   dcl stage2: Stage := new Stage("B001", 200);

   expecting(Stage`equal(stage1, stage2), true);
  );


  public static main: () ==> ()
 main() == (
   dcl t: TestStage := new TestStage();
   t.testSetName();
   t.testSetSeats();
   t.testEqual();
  );

end TestStage
```

Function or operation	Line	Coverage	Calls
main	27	100.0%	94
testEqual	19	100.0%	94
testSetName	3	100.0%	110
testSetSeats	11	100.0%	94
TestStage.vdmpp		100.0%	392

25 TestUtil

```
class TestUtil is subclass of TestClass

 operations


 private testDatePosition: () ==> ()
 testDatePosition() == (
 -- d1-d2 include d2-d4 fully
  dcl d1: Util`Date := mk_Util`Date(1997,02,14,20,55);
   dcl d2: Util`Date := mk_Util`Date(1997,02,14,21,55);

   dcl p1: Util`DatePosition, p2: Util`DatePosition, p3: Util`DatePosition;
   -- Position
   p1 := Util`getDatePosition(d1,d2);
   p2 := Util`getDatePosition(d2,d1);
   p3 := Util`getDatePosition(d1,d1);

   Util`expecting(p1,<Before>);
   Util`expecting(p2,<After>);
   Util`expecting(p3,<Same>);
 );
 private testDatesOverlap: () ==> ()
 testDatesOverlap() == (
  dcl d1: Util`Date := mk_Util`Date(1997,02,14,20,55); --- 1
  dcl d2: Util`Date := mk_Util`Date(1997,02,15,12,00); --- 2
  dcl d3: Util`Date := mk_Util`Date(1997,02,28,00,00); --- 3
  dcl d4: Util`Date := mk_Util`Date(1997,03,1,00,00);  --- 4

  dcl b1:bool, b2:bool, b3:bool, b4:bool, b5:bool, b6:bool, b7:bool;

  b1 := Util`datesOverlap(d1,d4,d2,d3);   -- Situation:  ---s1---s2---e2---e1   : overlap
  b2 := Util`datesOverlap(d1,d3,d2,d4);  -- Situation:  ---s1---s2---e1---e2   : overlap
  b3 := Util`datesOverlap(d1,d2,d3,d4);  -- Situation:  ---s1---e1---s2---e2   : no
  b4 := Util`datesOverlap(d1,d2,d2,d3);  -- Situation:  ---s1---e1s1---e2   : no
  b5 := Util`datesOverlap(d2,d3,d1,d4);  -- Situation:  ---s2---s1---e1---e2   : overlap
  b6 := Util`datesOverlap(d2,d4,d1,d3);  -- Situation:  ---s2---s1---e2---e1   : overlap
  b7 := Util`datesOverlap(d1,d2,d1,d2);  -- Situation:  ---s1s2---e1e2     : overlap

  Util`expecting(b1,true);
  Util`expecting(b2,true);
  Util`expecting(b3,false);
  Util`expecting(b4,false);
  Util`expecting(b5,true);
  Util`expecting(b6,true);
  Util`expecting(b7,true);
 );


 private testDaysOfMonth: () ==> ()
 testDaysOfMonth() == (
  dcl daysJan2000: nat1 := Util`DaysOfMonth(1, 2000);
  dcl daysApr2000: nat1 := Util`DaysOfMonth(4, 2000);
  dcl daysFeb2000: nat1 := Util`DaysOfMonth(2, 2000);
  dcl daysFeb1996: nat1 := Util`DaysOfMonth(2, 1996);
  dcl daysFeb1997: nat1 := Util`DaysOfMonth(2, 1997);

  expecting(daysJan2000, 31);
  expecting(daysApr2000, 30);
  expecting(daysFeb2000, 29);
  expecting(daysFeb1996, 29);
  expecting(daysFeb1997, 28);
 );


 public static main: () ==> ()
 main() == (
  dcl t: TestUtil := new TestUtil();
  t.testDatePosition();
 t.testDatesOverlap();
 t.testDaysOfMonth();
 );

end TestUtil
```

Function or operation	Line	Coverage	Calls
main	63	100.0%	8
testDatePosition	5	100.0%	8
testDatesOverlap	22	100.0%	8
testDaysOfMonth	48	100.0%	8
TestUtil.vdmpp		100.0%	32