Lab 4

December 11th, 2017

Schedule

Date	Time	Room
December 11th and 18th, 2017	11:30 - 13:00	Q117 and Q120
December 11th and 18th, 2017	13:15 - 14:45	Q117 and Q120
December 15th and 22nd, 2017	9:45 - 11:15	Q117 and Q120

Students assigned to Room Q120 - please, don't forget your notebooks!

Organisation

In this task, you can gain up to **4 bonus points** for the exam. The result of your team work depends on

- 50% functionality (2 points),
- 50% code quality (2 points).

The time in the lab is intended for asking questions concerning your implementations and not for solving all tasks: three hours of work will not be sufficient for solving this lab sheet. Read the following sections carefully on how to submit and how to earn full credit.

Functionality

Dependent on the functionality level of your implementation, you can get up to 2 points.

- 50% $\rightarrow$ 0.5 points
- 75% $\rightarrow$ 1.0 points
- 100% $\rightarrow$ 2.0 points

Check your code at least with the given test class!

Code quality

To reach the full amount of bonus points for code quality, mind the following requirements:

- "It works" is not enough!
- Avoid code duplication, create methods instead.
- Insert comments (if the behavior of your program can not be understood by reading the code).
- Format your code properly , take particular care of correct indentation.
- Use 'speaking' names (for variables, methods,...).
- Avoid 'magic numbers', use constants instead.
- Follow the recommendations given in the lecture as 'best practice'.

- Use appropriate return values and parameters.
- Use appropriate data types for variables.
- Test, test, test (and think about how and what to test).
- Create a user interface that is understandable and usable.
- **Plagiarism will count as a failed assignment!**

Keep in mind that points for code quality require a certain amount of code functionality. A well written and commented code with very little functionality can not reach the full amount of points for code quality!

Delivery

Read the following steps carefully before submitting your solution:

- Submission deadline: **December, 22nd, 23:59, CET**
- Submission via eMail to `lars.hanschke@tuhh.de` with `[DISS]` in subject line
- Send a ZIP-File containing
  - Entire project folder (of Eclipse project, not the entire workspace)
  - A *readme.txt* containing everything we need to know
    - Names and matriculation numbers of all three team members
    - Other things you need to tell us about your submission
- Make use of the *Export* function in Eclipse

Task description

Overview

The goal of this exercise is to develop an application to manage a high-bay harbor storage place containing a single rack with a given number of storage places (slots) of different sizes. Packets of different size can

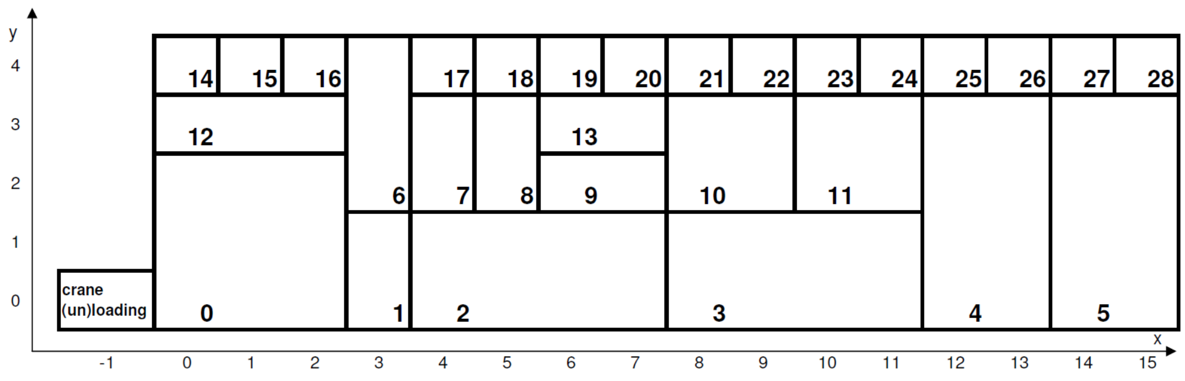

Figure 1: Structure of high-bay harbor storage rack

be stored and retrieved later on from the storage place by a crane.

The high-bay harbor storage consists of a number of slots, that are identified by their size, position and load capacity (max. supported weight). Figure 1 illustrates the example used in this lab. The crane allows to pick up packets at the loading position, move to the position of a slot, store the packet in it or remove the packet from the slot and transport it to the loading position. The crane is controlled with simple primitives; motors can be started to move it up, down, forward, backward or can be stopped.

Your first task is to implement a crane control that simplifies the usage of the crane. The control class enables the harbor storage management to store a packet at a defined slot position including picking it up at the loading position and storing it in the rack without controlling the motors of the crane directly. Packets can not be rotated to fit in a slot. The control class also allows retrieving a packet from a defined position and transporting it back to the loading position.

The next step is to implement the harbor storage management that determines and manages where (in which slot) each packet is stored and allows to find stored packets by their description. The storage management offers methods, which provide the functionality to store a packet dependent on the size and the weight of the packet. Packets might be too big or too heavy for certain slots in the rack; thus, the management has to find a suitable slot. Furthermore, the storage management offers the service to retrieve a packet later using its description.

As a last step, you add a user interface allowing users to control the behavior of your harbor storage management.

Tasks

You don't have to implement all classes, several classes are provided. Implement all missing classes and meet the requirements and fulfill the specifications given in this exercise sheet. Test your implementation and eliminate bugs. The work packages to be completed are:

- Implement class `CraneControl`. Write code to test it and perform your test.
- Implement classes `HarborStorageManagement`, `Slot` and `Packet`.
- Implement class `HarborStorageApp` containing the user interface.
- Test your software extensively via the user interface but also use and extend the provided test class `SimpleHarborStorageTest`.

A good idea is to split the work packages among your team members.

Before you start

Apart from this exercise sheet, make sure you also download the Zip-file *diss-lab4.zip* containing a complete Eclipse project from StudIP. It contains the classes we provide to you (see below) as well as skeletons of the classes you have to implement. (You may add more classes if this improves your solution.)

Step-by-step instructions:

- Download the file *diss-lab4.zip*
- Import the project into Eclipse:
  - Chose *File→Import...*

- Select *General→Existing Projects into Workspace* and click *Next*
- Check *Select archive file* and click on *Browse* to select *diss-lab4-template.zip*
- Check the project *diss-lab4-template* and click on *Finish*

Do not change package names or the structure of the project. After properly importing the project, you should see the package `de.tuhh.diss.harborstorage` containing several classes. You also see the file *harborstorage.jar* containing a compressed version of the simulator and other provided classes including `SimpleIO`. No errors should be reported by Eclipse, otherwise something went wrong.

Since the description of provided methods and classes on this sheet is only a summary, a detailed documentation of all provided files is given in *<project folder>/doc/index.html*. If you have problems understanding the functionalities of given methods, refer to the documentation first.

Harbor storage model

Figure 1 shows the structure of the harbor storage you are given along with this exercise. The size and the position of a storage place (slot) are given as dimensionless multiples of an elementary unit, which is also the basic unit of the coordinate system. The position in this harbor storage place is defined as the lower left corner of the storage place, e.g. slot number 11 has a size of 2x2x2 (the depth of the high-bay rack is set to a constant of 2) and position coordinates $x = 10, y = 2$. To be able to store or retrieve a packet correctly, the crane has to be exactly at the position of the storage place (lower left corner of slot). Please note that this definition of the slot position is just made for convenience when debugging – as your program queries all information (including the position coordinates).

Architecture of the high-bay harbor storage management application

The following presents an architectural overview of the high-bay harbor storage management application. Figure 2 shows the classes and interfaces used in it. Classes and interfaces that are marked gray are already implemented and have to be used.

You have to implement the other (non-gray) classes.

Note that return types and parameters for most methods and constructors are not given in Figure 2. They are mostly described later in this section or in the documentation given in the Eclipse project at `<project folder>/doc/index.html`. If no definitions are given, do your own adequate decisions. You may also add further methods and/or attributes or even classes if this improves the structure of your solution.

Class `PhysicalCrane` (provided)

Package: `de.tuhh.diss.harborstorage.sim`

This class provides control of the physical crane to store and retrieve packets to or from the harbor storage. It also displays a window that allows you to see the packets in the high-bay rack and the current crane location. The typical sequence of method calls to store a packet is as follows:

- Load a packet onto the crane by calling method `loadElement()`
- Drive the crane to the coordinates where the packet shall be stored using methods `forward()`, `backward()`, `up()`, `down()`, `getPositionX()`, `getPositionY()`, `stopX()`, `stopY()`.

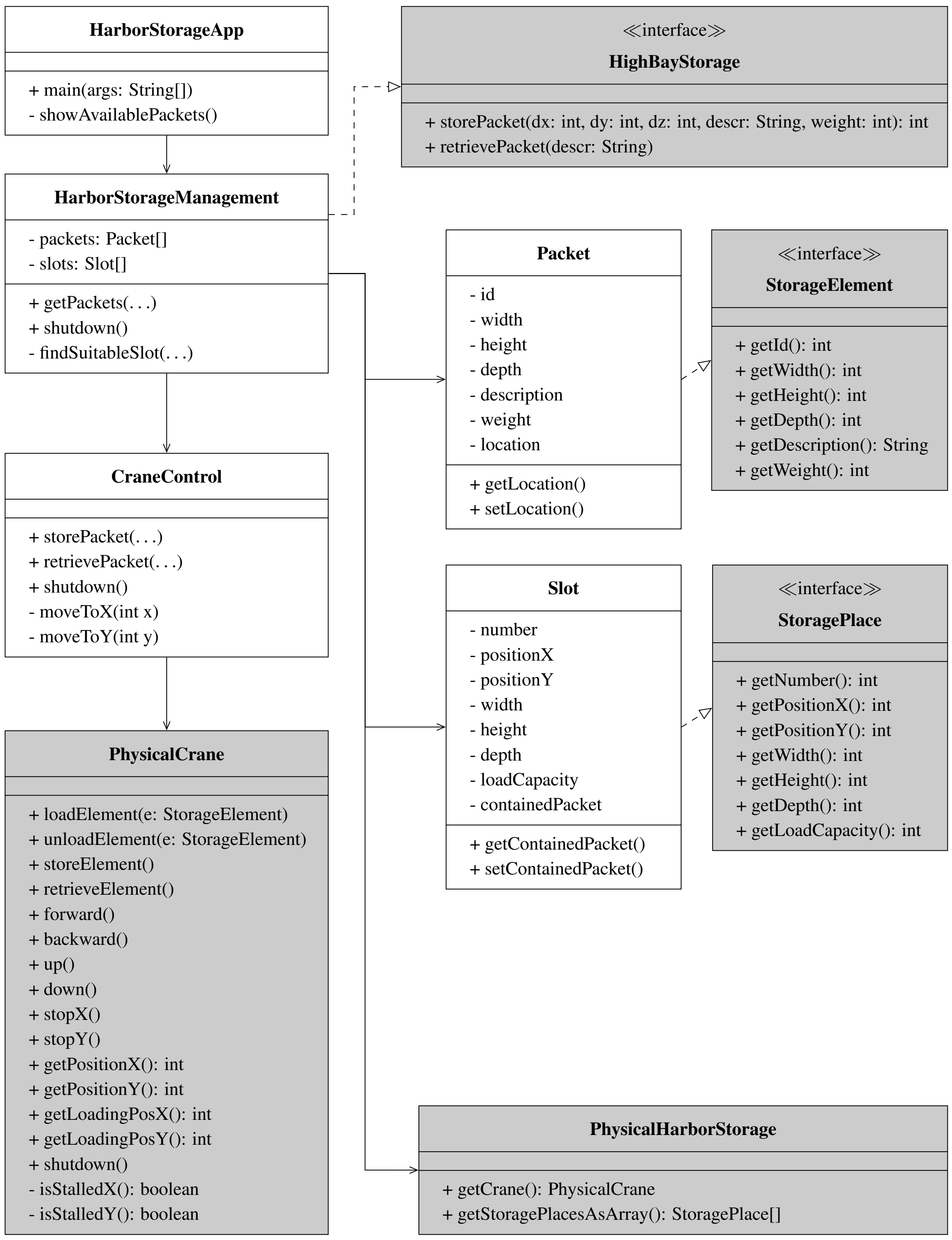

Figure 2: UML diagram of harbor storage

- Use method `storeElement()` to store the packet in the rack
- Drive back to the starting position

Before your program finishes its execution, you have to call method `shutdown()` to switch off the motors of the crane. Otherwise the simulation routine will not terminate.

Interface `StoragePlace` (provided)

Package: `de.tuhh.diss.harborstorage.sim`

This interface defines a set of methods that characterize a storage place (slot) within the harbor storage by providing methods to obtain its size, its number (= its ID) and its position. Method `getStoragePlacesAsArray()` of class `PhysicalHarborStorage` returns an array informing about available storage places.

Method summary:

- `int getWidth()` returns the width of a storage place (size w.r.t. to x-axis).
- `int getHeight()` returns the height of a storage place (size w.r.t. to y-axis).
- `int getDepth()` returns the depth of a storage place (size w.r.t. to z-axis).
- `int getNumber()` returns the identification number of a storage place.
- `int getPositionX()` returns the location of the slot w.r.t. the x-axis.
- `int getPositionY()` returns the location of the slot w.r.t. the y-axis.

Class `CraneControl` (to be implemented)

Package: `de.tuhh.diss.harborstorage`

You implement class `CraneControl` to simplify the use of the physical crane by providing methods to store and retrieve a packet. The class separates your harbor storage management from the task of controlling the physical crane by switching motors.

Constructor and method summary (add more private methods if needed):

- `CraneControl(PhysicalCrane cr)` constructs a `CraneControl` object that uses the physical crane given as parameter.
- `void storePacket(int x, int y, StorageElement packet)` controls the physical crane to drive to the loading position, to pick up the packet, to drive to the chosen slot at (x, y) and to store the packet.
- `StorageElement retrievePacket(int x, int y)` controls the physical crane to load the packet from slot (x, y) and unload it at the loading position.
- `void shutdown()` switches off all motors of the crane before terminating the program.

Hints: The crane can only move in x- or y-direction or can be stopped. To move the crane to a specific position x, implement a method `moveToX(int x)`. This method moves the crane `forward()` or `backward()` until the position x is reached. While moving, check the cranes position by calling `getPositionX()` until the final position is reached. Then use `stopX()` to abort crane movement. Implement the y-direction accordingly.

Interface `HighBayStorage` (provided)

Package: `de.tuhh.diss.harborstorage.sim`

This interface defines the most important methods to store and retrieve packets as implemented by class `HarborStorageManagement`. If packets can not be stored or retrieved, most methods shall throw an exception of class `StorageException` (Package: `de.tuhh.diss.harborstorage.sim`).

Method summary:

- `int storePacket(int x, int y, int z, String descr, int weight)` stores packets with given size, description and weight at a suitable slot in the high-bay rack and returns the packet's ID. Your method should find the best slot: the smallest slot with the smallest load capacity that is able to fit and carry the new packet.
- `void retrievePacket(String descr)` retrieves the packet with the given description. Check your harbor storage for the described packet, read its slot number and query the crane to retrieve a packet from the specified slot position. Delete the packet from the list of your packets and empty the slot in the high rack.

Class `HarborStorageManagement` (to be implemented)

Package: `de.tuhh.diss.harborstorage`

This class contains the central management logic of your application. It implements the interface `HighBayStorage`. In addition to the methods of this interface, the class provides a method to query information about all packets stored within the harbor storage.

Constructor and method summary (add more private methods if needed):

- `HarborStorageManagement()` constructs an object with an empty `PhysicalHarborStorage`, obtains its `PhysicalCrane` by calling `getCrane()` and constructs the `CraneControl`.
- All methods from interface `HighBayStorage` as described for the interface.
- `Packet[] getPackets()` returns an array with all packets stored in the rack. The array must not contain elements that are null or refer to the same packet.
- `int findSuitableSlot(Packet p)` iterates over the slots in the high rack and checks their suitability. A slot is suitable if it is empty, all sides are greater than the packet and a load capacity equal or greater than the weight. Since your HarborStorage should store as much weight as possible, so finding a slot with as little difference to the weight of the packet as possible is key. However, the first slot you find might not be the best one — check all slots before you decide for the most suitable.
- `void shutdown()` switches off all motors of the crane before terminating the program.

Hints: The two arrays of the class should/may be used as follows: `slots` stores information about all slots in the high rack (with ID, size, contained packet and load capacity) and `packets` is an array whose size is determined by the number of slots. Store each handed packet in `packets` and update the `containedPacket` attribute in `slots` when a suitable slot is found. Note you have to track the total number of stored packets or create method for obtaining the number to create a suitable return array for the method `getPackets()`.

Class `Packet` (to be implemented)

Package: `de.tuhh.diss.harborstorage`

This class abstracts a packet, which can be stored in and retrieved from the rack. It has to implement the interface `StorageElement` to allow its objects to be passed as parameter to method `loadElement()` of class `PhysicalCrane`. Apart from having an ID, a size and a weight, a packet contains a short textual description of its content. Also save the slot where the packet is stored in `location`.

Class `Slot` (to be implemented)

Package: `de.tuhh.diss.harborstorage`

This class is a model of a place in the harbor storage used to store a packet. It implements the interface `StoragePlace`. Apart from that, it holds the packet, which is currently stored.

Class `HarborStorageApp` (to be implemented)

Package: `de.tuhh.diss.harborstorage`

This class contains the user interface that allows the user to use the `HarborStorageManagement`. For that purpose it creates an object of class `HarborStorageManagement` and calls its methods. The class also contains the method `main()` as the entry point of the application. Also implement a method `showAvailablePackets()` that queries the harbor storage management class about its stored packets and prints a formatted list to the console to enable the user decide which packet to retrieve.

Hints: Split your application in several methods for creating dialogs with the user for: storing a packet, retrieving a packet and showing all available packets. Also implement methods to use when invalid choices of the user are entered. Use the provided `StorageException` which is thrown by methods of `HarborStorageManagement` to react correctly if packets can not be found.

User interface

The user interface implemented in class `HarborStorageApp` has the following properties. After starting the application, a textual menu is shown to the user that allows selecting one of the following tasks:

- Store a packet in the harbor storage
- Retrieve a packet from the harbor storage

To store a packet, the user has to provide the dimensions (width, height, depth), the weight and a textual description of the packet. A unique identifier for the packet is generated and the management application stores the packet in the physical harbor storage. If no empty and fitting slot can be found, the application needs to notify the user that the packet can not be stored. To retrieve a packet, the user enters the description of the packet. Then the packet is taken from its storage location. The user should be given an overview of the packets available in the harbor storage showing all attributes of the stored packets. The usage of your program for storing and retrieving two packets should be displayed similar to the following. Inputs of the user are shown in bold.

```
Size: 16x5
PhysicalCrane simulator started, v1.2

Welcome to TUHH/DISS Harbor Storage Management

*** Main Menu ***
0: Quit program
1: Store a packet in the highbaystorage
2: Retrieve a packet from the highbaystorage
Your choice: 1

*** Store a packet ***
Description: Packet 1
Width: 2
Height: 3
Depth: 1
Weight: 50
You entered packet "Packet 1" of size 2x3x1 and weight 50.
Shall we store the packet? (y/n) y
(crane <-- packet_1|size:2x3x1|Packet 1)
(slot_4|x12y0|2x4x2 <-- packet_1 size:2x3x1)
Packet stored in rack. The ID is 1

*** Main Menu ***
0: Quit program
1: Store a packet in the highbaystorage
2: Retrieve a packet from the highbaystorage
Your choice: 1

*** Store a packet ***
Description: Packet 2
Width: 1
Height: 1
Depth: 1
Weight: 10
You entered packet "Packet 2" of size 1x1x1 and weight 10.
Shall we store the packet? (y/n) y
(crane <-- packet_2|size:1x1x1|Packet 2)
(slot_14|x0y4|1x1x2 <-- packet_2 size:1x1x1)
Packet stored in rack. The ID is 2

*** Main Menu ***
0: Quit program
1: Store a packet in the highbaystorage
2: Retrieve a packet from the highbaystorage
Your choice: 2

Available packets:
1: Packet "Packet 1" Size: 2x3x1 Weight: 50
2: Packet "Packet 2" Size: 1x1x1 Weight: 10

*** Enter description of packet to be retrieved (0 = Abort) ***
Packet 2
(slot14|x0y4|1x1x2 --> packet: id=2 size:1x1x1)
(crane --> packet_2|size:1x1x1|Packet 2)
Packet retrieved
```

```
*** Main Menu ***
0: Quit program
1: Store a packet in the highbaystorage
2: Retrieve a packet from the highbaystorage
Your choice: 0

System ends.

PhysicalCrane was shut down.
```