Department of Informatics

Professor: R. Ingold
Teaching Assistants: M. Alberti, L. Vögtlin
Assistant: R. Geissbühler

Assignment 08

1 Theory

In this exercise you have to answer the following theoretical questions. Keep your answers short and precise.

(a) What is an exception in Java?

(b) What is the difference between `Exception` and `Error`?

(c) What is the difference between `RuntimeException` and `CheckedException` ? Make 3 examples for each category.

(d) Describe the two different solutions to handle a `CheckedException`. Additionally, elaborate on how do they differ in terms of responsibility (e.g. who has to take care of it) ?

(e) What is the keyword to use to "pay attention" when an exception is raised? What other keyword must be used jointly with it? Why?

(f) Is it possible to have multiple exceptions types in a `catch` clause?

(g) What is the keyword `finally` and how should it be used?

(h) What is the difference between the keywords `throws` and `throw`? Make a complete example for both.

(i) Given the following catch clause : `catch (Exception e)` what could you remark?

(j) When does it make sense to throw another exception inside a `try` block ? Make an example if appropriate.

(k) When does it make sense to throw another exception inside a `catch` block ? Make an example if appropriate.

(l) When does it make sense to throw another exception inside a `finally` block ? Make an example if appropriate.

(m) What is the advantage of the `try` blocks with resources? What property is a resource required to have to be used in this way?

(n) When is it useful to declare you own exception?

2 Implementation

For this exercise you are required to submit three files: Main.java, BankAccount.java and InsufficientFundsException.java, implemented as described below.

(a) Implement a `BankAccount` class with a `withdraw()` method which subtract a given amount to the current balance or throws a `InsufficientFundsException` in case the balance is insufficient. Define other methods and constructor as necessary.

(b) Implement a `InsufficientFundsException` class which extends from `Exception` and can be queried for the amount of the insufficient funds. `Hint: see first theory question`.

(c) Implement a `Main` class with a `main()` method which creates a `BankAccount` with initial founds of 500 and immediately withdraws 600. Handle the exception by printing to console the missing amount (in this case 100.0).

3 Debugging

In this exercise we want to practice the use of the debugger and familiarize with the error messages of the compiler. Refer to the files `Main.java` attached to this series for this exercise.

(a) Run the `Main` file.

(b) Notice that you can't because there is a compiler error in `bar()`. Report the `exact` compiler error.

(c) Describe what is the problem and explain why this is not happening in `foo()` although the code is extremely similar.

(d) Modify the code such that it compiles correctly.

For reference (you don't have to rewrite them yourself, we provide the java files attached to the series):

```java
package core;

import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        foo();
        bar();
    }

    static private void foo() {
        throw new IllegalStateException();
    }

    static private void bar() {
        throw new IOException();
    }
}
```

4 Bonus exercise

This exercise is taken from last year final exam. The estimated time for solving it was 10 minutes. Try to solve it first in a “exam-like” setting — no internet, no cheating — and only after fix it (if necessary) by using all resources at your disposal.

```java
1 public class Main {
2     public static void main(String[] args) {
3         BufferedReader br = new BufferedReader(new
             FileReader("myFile.txt"));
4     }
5 }
```

Fr Lorsqu’on compile le programme ci-dessus, on obtient le message d’erreur ci-dessous.
De Wenn wir den obenstehenden Code kompilieren, wird der folgende Fehler geworfen:

```
Error:(3, 48) java: unreported exception
    java.io.FileNotFoundException; must be caught or declared to be
    thrown
```

Fr (a) Que signifie le message ?
De Was bedeutet diese Nachricht?

....

....

....

....

....

Fr Proposez deux approches différentes pour corriger cette erreur en complétant les codes ci-dessous (faites attention aux accolades).

De Geben Sie zwei verschiedene Ansätze, um diesen Fehler zu verhindern, indem sie den untensteheneden Code kompletieren. (Seien Sie vorsichtig mit den geschweiften Klammern.)

(b) Solution 1:

```
public class Main {


    public static void main(String[] args)




            BufferedReader br = new BufferedReader(new
               FileReader("myFile.txt"));







    }


}
```

(c) Solution 2:

```
public class Main {


    public static void main(String[] args)




            BufferedReader br = new BufferedReader(new
               FileReader("myFile.txt"));







    }


}
```

5 Project

As our Game Objects can now shoot you need to make the relevant entities able to get hit. For this you need a bounding box for each object and to check for collision.

5.1 Create a Bounding Box

The bounding box represents the border enclosing the visualization of a Game Object (see figure 1). So each of your Game objects needs to have a method which returns its bounding box.

Create in the appropriate class/es a method `getBoundingBox()` which returns a `java.awt.Rectangle`[1]. You do not have to draw the bounding boxes onto the canvas!

Hint: You can use the x, y, width, height constructor of the `Recangle`. To get the width and height you can use the `.getWidth(null)` and `.getHeight(null)` methods from the first series.

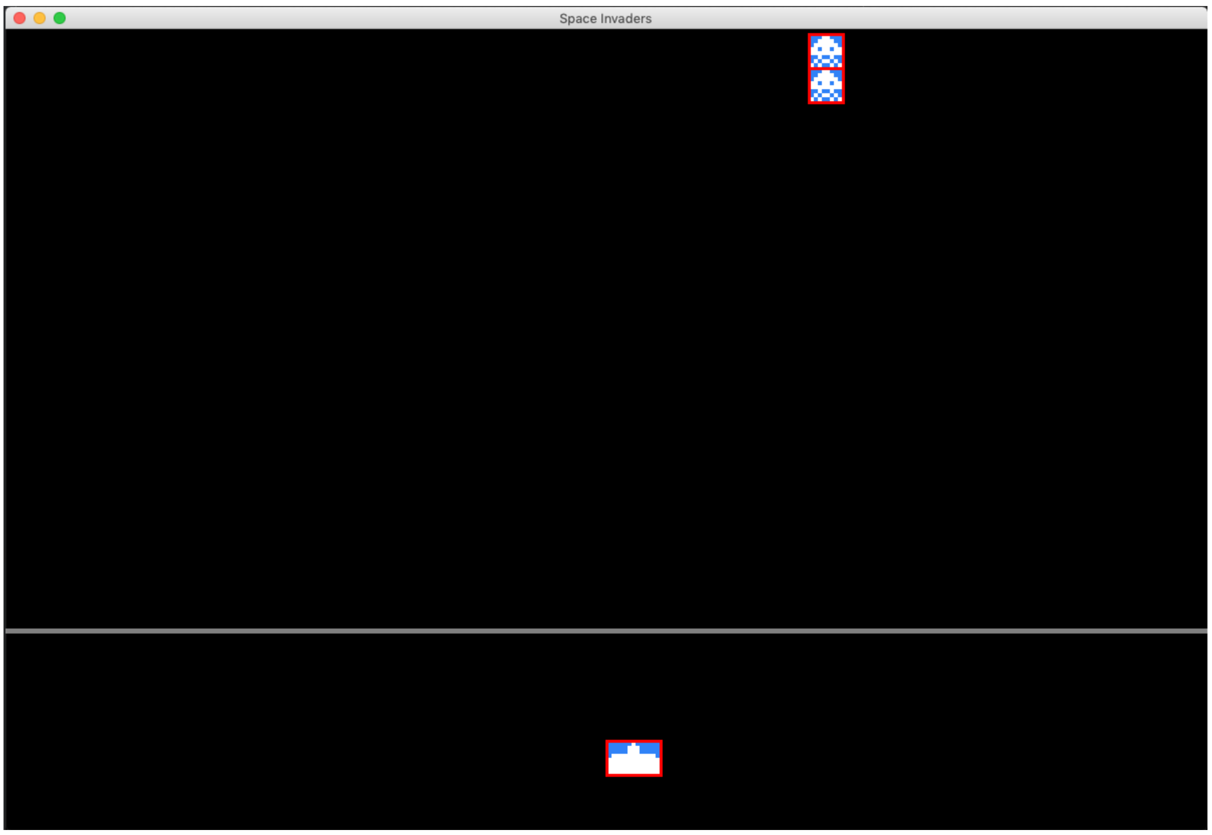

Figure 1: In red you see the bounding boxes of the Game Objects.

[1] https://docs.oracle.com/javase/8/docs/api/java/awt/Rectangle.html

5.2 Check for Collisions

To see if two Game Objects collide you have to check if the bounding boxes of the objects intersect. As we are using `java.awt.Rectangle` as return of the `getBoundingBox()` method this is easy to detect.
Create in the class `Sprite` a method `checkCollision()` which takes as an argument a sprite and returns a boolean. The method returns `true` if the two sprites, so the input sprite and the sprite on which we are calling the method, are colliding and `false` if they do not.
Now you need to check if the bounding box of the input sprite is intersecting with the bounding box of the current objects bounding box. This is done by calling the method `intersects()` on one of them and hand over the other object as parameter to the method. Return now the output of the `intersects()` method.

5.3 Adapt ShotList and EnemyList

To make your life easier when implementing the collision checks you have to adapt the EnemyList and the ShotList. You have to implement these three methods in both classes:

isEmpty(): The purpose of this public method is to tell to user if the list is empty or not. If the list is empty, so the size of your list is smaller or equal to 0, it returns `true` and `false` otherwise.

findIndex(Enemy enemy): With this private method we find the index of an object in the elements array. You have to iterate over all elements in the `elements` array (i ¡ size) end check if the given object matches one of the elements in the array. If this is the case return the index, else return -1.

remove(Enemy enemy): (change signature for ShotList) This method will make it possible to delete given object in the list. You can just copy the code from the method `remove(int i)` and get at the beginning the index of the object with the help of the method `findIndex()`. If `findIndex()` returns -1 you have to stop the execution of the method else you can continue as in the `remove(int i)` method.

5.4 Extend the SpaceInvaders class

As you now have the collision detection you can start using it in the game. You have to create a new method `resolveShotsCollisions()` which you have to call after you moved the enemies in the `step()` method. Implement the method as shown in the following pseudo-code:

```
ShotList shotsToRemove
EnemyList enemiesToRemove
// Check for damage
for shot in shots:
    // Check if enemies are hit
    if shot == null:
        continue

    if shot.direction.equals("UP"):
        for enemy in enemies:
            if enemy.checkCollision(shot):
                enemy.gotHit(shot)
                shotsToRemove.add(shot)
                break

    // Out of screen shots will be removed
    if shot.getY() < 0 || shot.getY() > BOARD_HEIGHT:
        shotsToRemove.add(shot)

//remove all shots
for shot in shotsToRemove:
    shots.remove(shot)

// Check for dead enemies
for enemy in enemies:
    if enemy.isDead():
        enemiesToRemove.add(enemy);

// remove dead enemies
for enemy in enemiesToRemove:
    enemies.remove(enemy);
```

Now you should be able to shot with your player the enemies. But what happens when all enemies are dead (enemy list is empty) from a technical (Java) and a non-technical (game/user) point of view? Fix this problem by printing out a message to the console of call the appropriate method in the `board` class.

5.5 Exceptions

We already added some exceptions for you in the code but there are still some places where we would need them. Throw exceptions where appropriate but at least at the following positions:

1. in the classes ShotList, EnemyList and IntegerList if we call the remove function but our list is empty (hint: NegativeArraySizeException)
2. in the classes Enemy and Player if they got hit (gotHit()) when they are already dead (healthPoints ≤ 0)