1 Battleship game

In this exercise you will implement a simplified version of the widely known game Battleship. We will not use the judge for this exercise - this allows us to provide you a bigger skeleton project which can be found here `http://lec.inf.ethz.ch/baug/informatik2/2015/ex/Assignment06.zip`.
For feedback concerning your exercise please mail your TA's. (which is always an option - also for the other exercises)

2 Setting up the project

Download and extract the skeleton from the link above. In Eclipse select File - Import - General - Existing Projects into Workspace. Navigate to the directory in which you extracted the downloaded code and select the Assignment06 project. Once finished you should see the project in the package explorer within eclipse. See figure 2 figure 3 and figure 4.

2.1 Inspect the code

The code is packaged into three subpackages. In a real world project you might not package into that many packages. Part of the exercise is to get you used to work with packages, which is the main motivation for this organization.

3 Draw shapes

Your first task is to implement the drawList method in ShapeList.java

```
//pre: -
//post: draw all the objects contained in the list
public void drawList(ImageViewer panel)
{
    //replace me!
}
```

Simply traverse the list and for each polygon contained in it, call its draw method. You can test your implementation by executing the main method of TestDrawShapes.java

4 Random placement

Now that we are able to draw ships we will focus on placing ships randomly on the game field.

4.1 Random placement with collisions

To get you started, you first have to implement

```
//pre: -
//post: insert nr_ships many polygons representing ships into the hidden_ships list
//      use the "createRandomShipConfig" routine to get a random configuration for your
//      ship. watch out that config[2] is the direction
//      (used first in Shapes.Ship method)
private void placeRandomShipswCollision(int max_x, int max_y, int nr_ships){ ... }
```

"Ship polygons" can be created with the static method $Shapes.Ship$. You can test your implementation by running the main routine of TestPlacementwithCollision.

4.2 Random placement with collision detection

We would like ships not to overlap. To achieve this we implement a very simple collision detection.

4.2.1 Collision detection

In 1 we sketch a very simple collision algorithm. Everytime checkCollision is called, it will check if the square described by the coordinates $newx$ and $newy$ of one corner and equal side length $Shapes.max_size$ overlaps with any square described by $placementx[i]$ and $placementy[i]$ also with size $Shapes.max_size$. When you check for overlap, be aware that in case of the default value an element of $placementx[i]/placementy[i]$ might be -100.

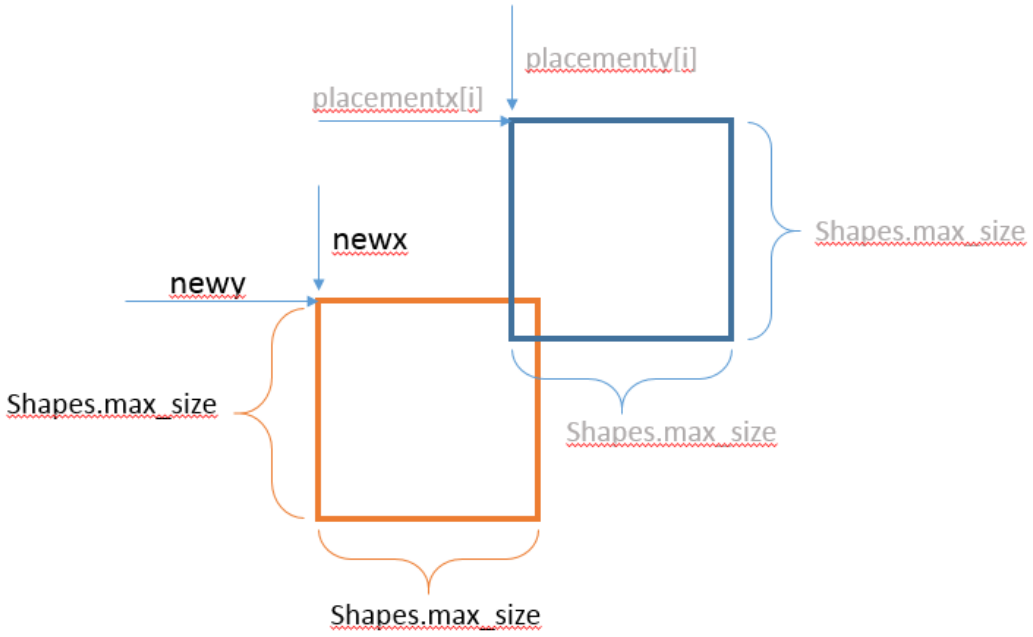


Figure 1: sketch of the simple collision detection

```
//pre: placementx and placementy hold the info of where ships have been placed so far
//     if ships have not been placed yet the according array element will contain -100
//     as a default value
//post: true if the square described at position newx,newy (+ Shapes.max_size) does
//      collide with any box described by placementx[i], placementy[i]
//     (+ Shapes.max_size)
private boolean checkCollision(int[] placementx, int[] placementy, int newx, int newy)
{ ...}
```

4.2.2 Placing a ship with collision detection

$placeRandom$ is supposed to position a single ship. To achieve this create a random config - check if this config would overlap with any existing ships. If it overlaps just retry till you succeed in finding a non overlapping config. Once such a config has been found add your placement to $placementx[nr]$ and $placementy[nr]$ and return a ship polygon with the according config.

```
//pre: placementx and placementy hold the info of where ships have been placed so far
//      if ships have not been placed yet the according array element will contain -100
//      as a default value
//      parameter nr just tells the number of the ship we currently creating
//     max_x and max_y are used for the random Config
//post: added the non overlapping config to placementx[nr] and placementy[nr]
//        return a Shapes.Ship with the according config
private Polygon placeRandom(int[] placementx, int[] placementy, int nr,
        int max_x, int max_y) { ...}
```

4.2.3 Creating multiple random ships with collision detection

Finally your task is to complete the function:

```
//pre:  max_x and max_y are used for the random config
//      nr_ships is self explanatory
//post: insert nr_ships many ship polygons into the list "hidden_ships"
private void placeRandomShips(int max_x, int max_y, int nr_ships) {
```

Create nr_ships many ship polygons that are not colliding using what you implemented so far. Everytime you create a ship, insert into the list $hidden\_ships$. You can test your implementation by calling the main method of TestRandomPlacement.java.

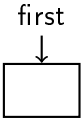
5 Game logic

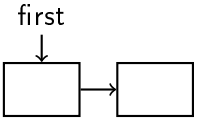
In the last section of this project we implement the game logic of Battleship. The essential part is that we need to be able to detect if a "shot" hit a ship on the play field.

5.1 Detecting a hit and removal from list

We implement detection and removal in a single operation. You need to traverse the list and for each element check if calling $PointInPolygon(x, y)$ for the polygon contained in the node returns true. If so - you are supposed to remove the node from the list and return the polygon. If you do not find any node for which this is true, return null. For removing a node from a list you need to be aware of three scenarios (empty list, only one element, more then one element):

- 

- 

- 

```
//pre: x and y describe the coordinate for which we would like to check
//     each list element - if the coordinates are contained in the polygon
//     described in the nodes
//post: if we find a node, which contains the point,
//      remove the node from the list and
//      return the polygon that was removed
//      if we dont find a node - return null
public Polygon remove(int x, int y) { ...}
```

5.2 Shooting at a position

```
The final piece of code is to complete:
//pre: x,y describe the coordinates you want to fire a shot at
//post: always draw a shot at the given coordinates by adding a
//      Shapes.Shot to the shots list.
//      try to remove from hidden_ships list with the remove function you just
//      implemented.
//      if remove returns a polygon (not null) -> insert it into the hit_ships list
//      and return true
//      otherwise return false
public boolean shotPosition(int x, int y) { ...}
```

If you correctly implement shotPosition you can test the whole game by running the main method of "TestSimulatedPlayer.java" - which will randomly fire at the game field till it hit all ships. Alternatively you can play the game yourself by executing the main routine of Game.java.

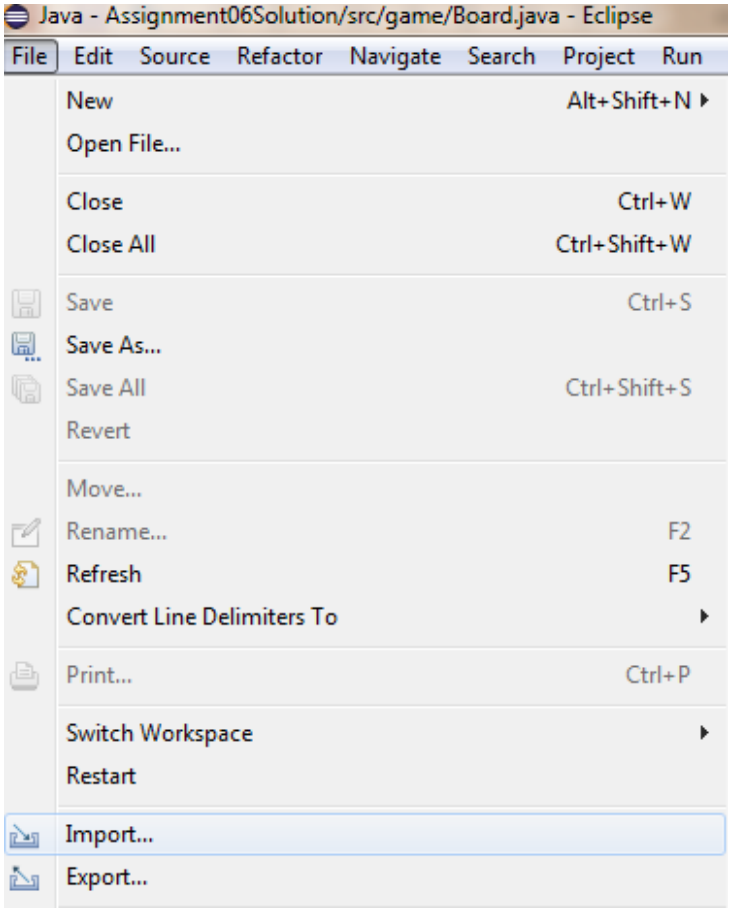


Figure 2: Setting up the project step1

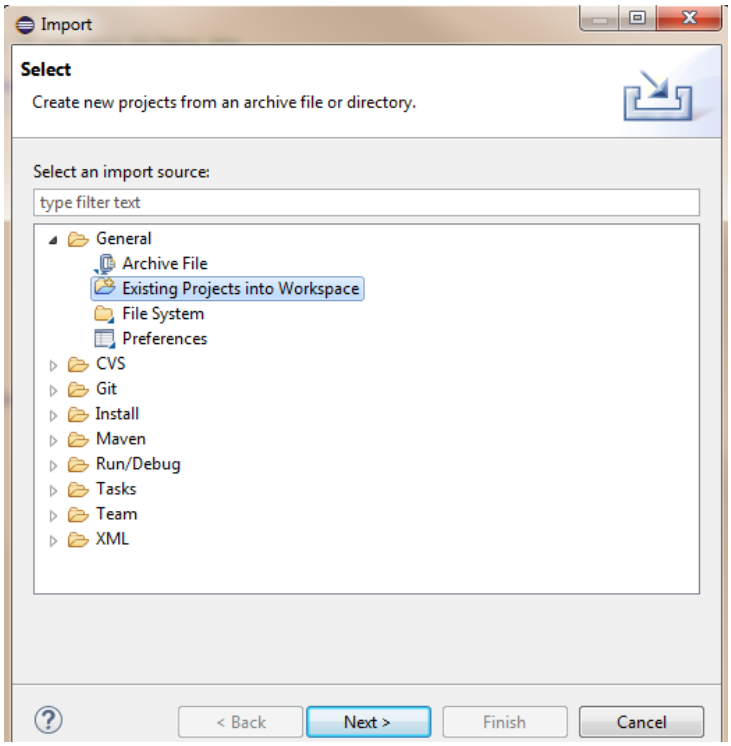


Figure 3: Setting up the project step 2

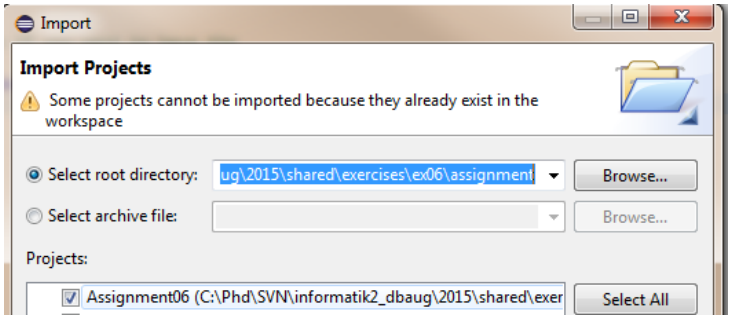


Figure 4: Setting up the project step 3