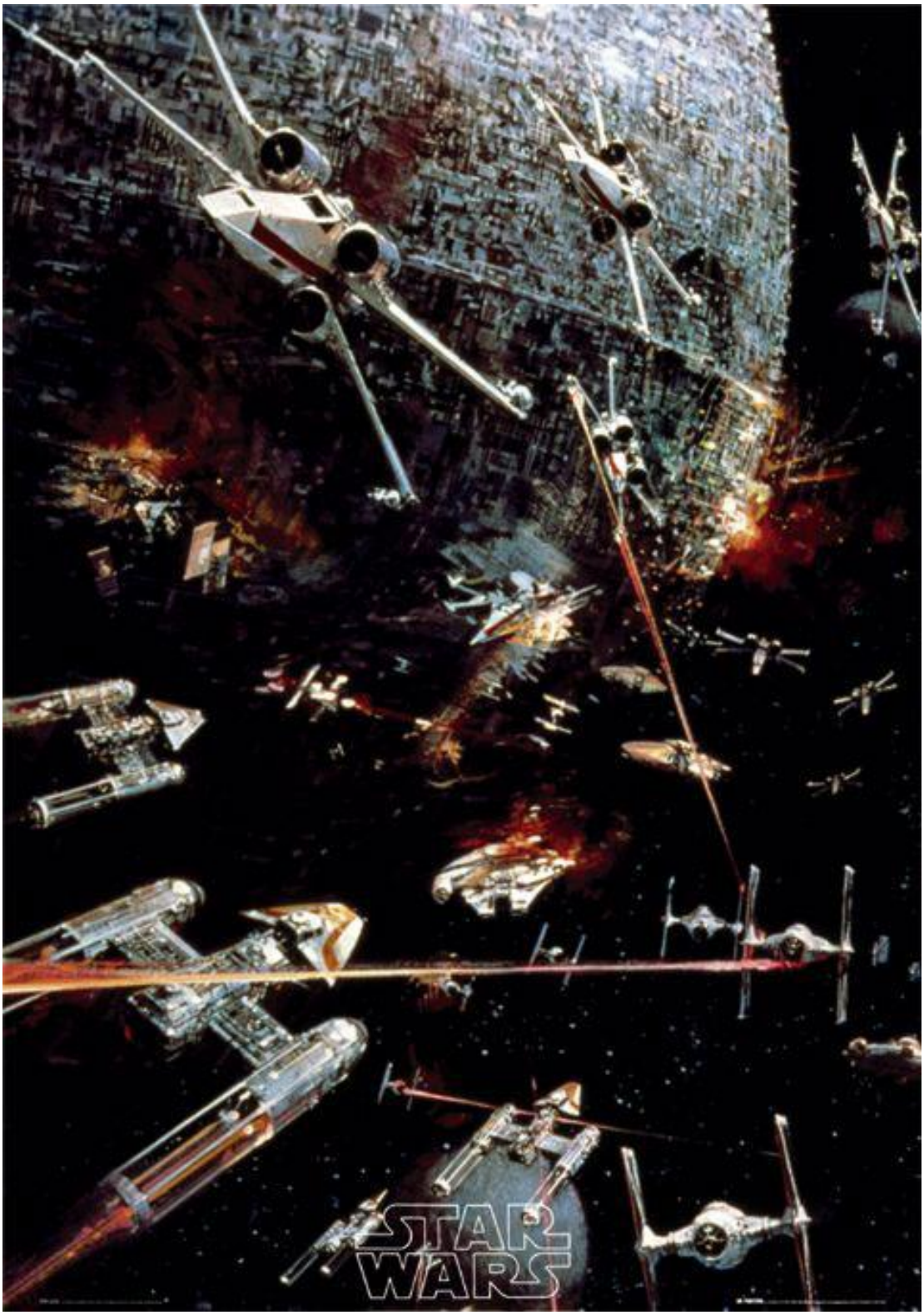
Intro Ex9 - Space Wars



Objectives

The purpose of this exercise is to practice object-oriented design and, in particular, to understand how to make use of inheritance and polymorphism within a program.

General

In this exercise, you're allowed to work in pairs, which can be mixed between courses (for example one student from 67101 and the other from 67130). You can find your significant other using the dedicated forum.

This exercise was used in several times in the past, but you should resist the temptation to use someone else's solution from a previous year. This would be considered cheating, and we will compare your code against all submissions from the last 5 years.

Students that repeat the course and did this exercise before, can base their solution on their own code from the previous time (and say so in the README), however, in any case such students are NOT allowed to pair up with someone who is taking the course for the first time.

Please submit your solution from the user with the lexicographically lower **username**. (e.g. if the pair is benni78 and moshe15 - submit from benni78). Instead of the usual grading procedure, in this exercise you will have a face-to-face interview with a grader, where you will be asked various questions about your design. You are very strongly encouraged to work on everything together, and not divide work between the pair (let alone divide work asymmetrically). It is typically very easy for a grader during an interview to identify if one of the partners was "dominant" while the other was more "passive". See CS submission guidelines.

Exercise Description

In this exercise you will design and implement parts of the SpaceWars computer game. We provide you with the driver class that controls the game (in SpaceWars.java), as well as a class that manages the physics of the spaceships (the class SpaceShipPhysics in the intro package, see its API).

Your job is to design and implement the different SpaceShips that take part in the game according to the rules outlined below.

The SpaceWars class contains the main method and can be activated in the following manner:

```
java SpaceWars <space_ship_type> <space_ship_type> [<space_ship_type> ...]
```

where the spaceship types are one letter arguments that denote the spaceships that will be used in the game:

- h - Human controlled ship (controlled by the user).
- c - Crazy human controlled ship (controlled by the user).
- f - Floater: a ship that floats around and does nothing.
- r – Runner: a ship that tries to avoid all other ships. Has a spying ability that will be explained later.
- b - Basher: a ship that deliberately tries to collide with other ships.
- a - Aggressive: a ship that tries to pursue and fire at other ships.
- s - Special: a ship with some interesting behavior that you will define yourselves (you should explain its behavior in the README).

Your job is to implement all of the ship types above and add them to the game. See more details on the spaceships' behavior below.

Provided classes:

You can download the entire code of this exercise (except the school solution) from here.
In order to use the relevant classes we provide for this exercise you should import the following packages at the beginning of your java files:

```
import il.ac.huji.cs.intro.ex9.*;
import java.awt.Image;
```

- SpaceWars - This file is the driver for the SpaceWars game. It is provided by us. You can't change this class, since you should not include it in your submission. All the code that you write in this exercise should compile with this driver as is. The SpaceWars class runs and manages the computer game. In order to manage the different spaceships in the game it holds an array of objects of type SpaceShip.
- SpaceShipFactory - This class has a single static method (`createSpaceships(String[])`) that is used by the driver to create all the spaceships objects according to their type. You will need to implement this static method (Since your implementation of the spaceship isn't written yet, we do not know how to construct the different spaceships in our driver, so you will need to do this yourselves in the above `createSpaceships` method).
- SpaceShip - We are providing you with the API every spaceship should implement. In your exercise solution, you must have a file named SpaceShip.java, but it is your choice whether this file should define an interface, an abstract class, a base class for all the spaceships or some other option. You may also add methods to this API but you cannot remove any.

Your job is to write the code for the SpaceShips. You can also add additional classes and files to the program. Your program should make use of polymorphism for the different spaceship types. See some tips for the suggested design below.

Other than that, it is subject to your own design decisions, which you should explain in the README, but you should remember to follow what you have learnt and your code will be graded accordingly.

Provided APIs

- SpaceShipPhysics - This class represents the physical properties of a spaceship (its position, heading, speed and so on). Every spaceship needs to have exactly one such object. When created (through the default constructor) the object is initialized randomly. A SpaceShipPhysics object should be created (only) when a spaceship is created, after it is reset (because it died), or if it has teleported. All the updates of the spaceship's position should be done through this object. See the API of this class.
- GameGUI - This class is used by the driver to draw the spaceships on the screen, and should be used to get input from the user. One of the methods each spaceship has is a method that returns a java.awt.Image that represents the image of the spaceship that needs to be drawn to the screen. There are four such images, and they are defined as constants of the GameGUI class. Two images represent a human controlled ship and a computer-controlled ship, and two additional images represent the same ships with their shields activated. In addition, this class has methods that detect if the user has pressed certain keys on the keyboard. These methods should be used by your program to get input from the user and control the human-controlled spaceship. See the API of this class.

The Rules of the Game

These are the rules that your implementations of the spaceships should conform to:

All of the spaceships in the game have the following attributes:

- A SpaceShipPhysics object that represents the position, heading and velocity of the ship.
- An Energy level.
- A Health level.

The health of a spaceship starts at 10, and is reduced by 1 every time the ship is hit by a shot, or collides with another spaceship. If the spaceship has its shield turned on, then there is no reduction of health. The energy level of a spaceship starts at 200. A spaceship recharges 1 energy unit every turn (up to the maximal value of 200). A spaceship loses energy when it fires (25 units per shot) teleports (150 units) or uses its shield (3 units per round in which the shield is active). The energy level can never go below 0, and if one of the above actions is attempted when there is not enough energy to carry it out, the action should not take place.

The game proceeds in rounds, in which each of the spaceships can do the following actions (each action should happen at most once per round, and in a given round any subset of these actions can be performed):

- Accelerate: the spaceship will accelerate a bit in the direction it is facing.
- Turn: the spaceship can turn to the left or to the right slightly.

Both the turning and acceleration actions should be done through the SpaceShipPhysics object, using a single call to the move() method. This method should be called exactly once per round even if the spaceship does not accelerate or turn.

- Teleport: The spaceship will disappear and reappear at a random location. This should be done by creating a new randomly generated SpaceShipPhysics object for this ship.
- Fire a single shot: the spaceship can fire a single shot (this can be done by calling the addShot() method of the SpaceWars driver). After firing, the ship's guns cannot be used for a period of 8 rounds. After this time has passed, the spaceship may fire again, but only if it has the necessary 25 units of energy.
- Turn on its shield (for this round only). If this action takes place, then the spaceship cannot be damaged by a shot or a collision during this round.

Since several actions can take place in each round, the actions in the game must take place in the following order:

1. The Teleport action should occur first in the round (if there is enough energy and it was attempted).
2. The accelerate and turn actions should occur at the same time using a single call to the move() method of the SpaceShipPhysics object.
3. The shield should be raised (if there is enough energy, and the action was attempted).
4. A shot is fired (again only if it was attempted, and the ship can indeed fire).
5. The ship regenerates 1 unit of energy for this round.

When a ship dies (because its health reaches 0 or lower) the SpaceWars driver will call its reset() method. The spaceship should then reappear in a new random position, with full health, and energy, as if it was only now created. The SpaceWars object will automatically keep track of the number of times each ship was destroyed. The game goes on indefinitely, until the escape key is pressed, or the window is closed (this functionality is already built in).

The Different Types of SpaceShips:

- Human controlled - This spaceship is controlled by the user. The keys used to control the spaceship are: left-arrow, right-arrow, up-arrow to turn left, right and accelerate. 's' to fire a shot. 'd' to turn on the shield. 't' to teleport.
- Crazy Human - The same as human controlled but in addition, in each turn there is a 2 percent chance that it will try to teleport.

There should be only one human controlled ship in a game (either crazy or normal).

- Floater: This spaceship performs no actions at all. It floats around at a constant initial speed and neither turns nor accelerates.
- Runner: This spaceship attempts to run away from the fight. It will constantly accelerate, and turn away from the nearest ship. The runner has the spying ability and will attempt spying on each round. It will try to get the nearest ship cannon angle from himself. If that angle is smaller than 0.2 radians (in any direction) and the distance from the nearest ship is smaller than 0.2 units, the runner will try to teleport.
- Basher: This ship attempts to collide with other ships. It will always accelerate, and turn towards the closest ship. If it gets within a distance of 0.2 units from another ship, it will turn on its shield.
- Aggressive: This ship pursues other ships and tries to fire at them. It will always accelerate, and turn towards the nearest ship. If its angle from the nearest ship is 0.2 radians or less (in any direction) then it will open fire.
- Special: You should think of another ship that has some interesting behavior and makes the game more fun.

Methods to get the distance or angle from another spaceship are part of the SpaceShipPhysics class. The closest spaceship can be found using the getClosestShipTo() method of the SpaceWars game driver.

Design and Implementation Hints

We suggest that you use one of the following designs for your program (though you do not have to). Since all spaceships have the same possible actions, you might want to add to the SpaceShip API methods that correspond to the actions (fire, teleport etc.). The difference between the spaceships is only in their behavior during each round. Here are two of the possible solutions that can be applied:

1. Create the base class SpaceShip that implements the SpaceShip API and several subclasses of it, each of which overrides the implementation of the doAction() method of the spaceship. This will lead to a different behavior for each of the subclasses.
2. Create a spaceship object (implements the SpaceShip API) that will be used to represent all the spaceships. Each spaceship object will contain inside it another object that will act as its "brain", i.e., an object that will control the behavior of the spaceship. To do so, you can define a 'Behavior' interface that has a single method that does the action for the spaceship. Then, define several classes that implement the Behavior interface – one for each of the spaceship types in the game, and use one of them inside the implementation of the spaceship (in a polymorphic way). This design is slightly more flexible. It relies on composition of objects and therefore it is easier to change the behavior of a spaceship even after its creation.

Testing and Grading

This exercise will not be checked using automatic tests. The graders will go over your code, play your game and interview you about your design. Since we do not provide you any unit tests, it is your responsibility to write them. You are required to write basic unit tests for your spaceships. These tests won't be checked against classes with bugs but their code will be reviewed by the graders. We do not

expect your unit tests to be comprehensive or to test things that are difficult to test, just to help you develop your code with the confidence that it is reasonably correct.

README

In the README file you should explain your design decisions and describe the behavior of the special ship, and any other comment about this exercise.

AUTHORS

Pay close attention to these instructions because failing to follow them will result in point reduction!
You should submit a file called AUTHORS with a single line of the form >> login1,login2. e.g. benni,moshe. See AUTHORS example. Remember you should submit from the user with the lexicographically lower username. If you accidentally submitted from the wrong user account, re-submit again from the right account and upload another jar from the wrong user, which contains only a README file that says you made this mistake.

School Solution

You can try out the school solution by typing: `~introcsp/bin/ex9/SpaceWars <two_or_more_spaceship_types>`

Javadoc

Document your code using the javadoc documentation style, as described in the coding style guidelines.
You can create html documentation files by invoking: `javadoc -private -d doc *.java.`
This command should succeed without any warnings or errors. You can view the html files (created in the "doc" directory) using a web browser. Given that this exercise is a design one, we will emphasize proper API documentation.

Submission and Further Guidelines

Your solution should include the following files:

- SpaceShip.java
- SpaceShipFactory.java
- unit tests
- AUTHORS
- README
- Any other java file needed to compile the program (but NOT SpaceWars.java)

The layout of the files should be as described in the course coding guidelines.

Do not submit any html files or class files.

When executing: `javac *.java` your files should compile with our driver without any warnings or errors.

Good Luck!!!