CS102 – Algorithms and Programming II
Programming Assignment 3
Spring 2024

ATTENTION:

- Compress all of the Java program source files (.java) files into a single zip file.
- The name of the zip file should follow the below convention:
  CS102_Sec1_Asgn3_YourSurname_YourName.zip
- Replace the variables "Sec1", "YourSurname" and "YourName" with your actual section, surname, and name.
- You may ask questions on Moodle and during your section's lab.
- Upload the above zip file to Moodle by the deadline (if not, significant points will be taken off). You will get a chance to update and improve your solution by consulting with the TAs and tutors during your section's lab.

GRADING WARNING:

- Please read the grading criteria provided on Moodle. The work must be done individually. Code sharing is strictly forbidden. We are using sophisticated tools to check the code similarities. The Honor Code specifies what you can and cannot do. Breaking the rules will result in disciplinary action.

Robot Battle Simulation

For this assignment, you will implement a console application to simulate turn-based combat between two teams of robot warriors. You are expected to take full advantage of polymorphism and inheritance in Java, keeping the common functionality in the parent classes with required modifications done through overridden methods.

Include a Simulation class with a `void initialize(int teamSize)` method to create two teams of robots of the given size. The two teams, red and blue, will have an equal number of robots initially. The robot types in a team will be determined randomly; there can be zero or more of each type of robot in the team. After creating all the robots of a team, sort them based on their speed values such that the ones with greater speed value appear first in that team's robot array.

Include a `void simulate()` method to simulate the battle between two teams. This method should initially determine which team starts first. To this end, calculate the sum of the speed values of the robots of each team; the team whose speed sum is higher starts first. If the speed sums are the same, the red team starts first. The robots of each team take turns to perform their attack. Suppose the red team starts first, and we have 5 robots; the order of attack would be as follows based on team and index: red 0, blue 0, red 1, blue 1, red 2, blue 2, red 3, blue 3, red 4, blue 4, red 0, blue 0, … Note that the order of the robots within the team is based on speed, so the robots that act first have higher speed.

Robots can be destroyed when attacked by the opposite team's robots. If a robot at a specific index is destroyed, remove it from the array and shift the remaining ones accordingly. Suppose each team started with 5 robots as given in the table below:

0	1	2	3	4
A	B	C	D	E

If robot C is destroyed, the modified array would be:

0	1	2	3
A	B	D	E

You are free to use ArrayList of Robot objects to handle teams. The simulation continues until one of the teams has all of its robots destroyed. Teams still take turns if the number of remaining robots is unbalanced. Suppose the red team has 1 robot and the blue team has 3 robots; then the order of performing attack would be as follows based on team and index: red 0, blue 0, red 0, blue 1, red 0, blue 2, red 0, blue 0, red 0, blue 1, …

The simulation should display meaningful messages to describe what is going on; i. e. who is attacking who with what kind of attack.

Implement an abstract Robot class to define the top-level logic of the robots. This class should include the robot's health, attack, and speed as double variables, a string name, and a boolean isRedTeam to specify the team of the robot. Also include the following methods:

- `abstract void attack(Simulation s)`: This method will handle the robot's attack and needs access to the Simulation class to choose a target opponent. Different robots prefer different targets, and subclasses of Robot will override this method based on their specifications.
- `boolean getHitAndIsDestroyed(double damage)`: This method will handle receiving damage for the robot. You will subtract the given damage from the robot's health; if the health becomes zero or less, the robot is destroyed. Return true if the robot is destroyed after receiving the damage; return false otherwise.

We have the following subclasses of Robot and the corresponding stats:

Class Name	SimpleBot	PredatorBot	DefenceBot	SpeedBot	SpreadBot	OneBot
Code	S	P	D	X	K	O
Health Range	2-3	2-3	3-6	1-2	2-3	0.5-1
Attack Range	1-2	2-3	0.5-1	1-2	0.5-1	4-5
Speed Range	1-2	0.5-1	0.5-1	3-4	0.5-1.5	0.5-1

The constructors of the subclasses of the Robot class determine the health, attack and speed of the robot instances by random variables within different ranges. For example, the constructor of SimpleBot should determine its health as a random double in the range [2,3). Consequently, each instance of the same class would give slightly different values; for example, one SimpleBoy can have a health of 2.2 and a different one of 2.6, and so on.

Each robot is assigned a name based on its code and production order. For example, the first robot created for the simulation would have production number 0. Suppose the constructed robots are SimpleBot, SimpleBot, DefenceBot, OneBot, and OneBot; then the corresponding names would be: S0, S1, D2, O3, and O4. Production numbers are not affected by the team, suppose we generate teams of 5 robots, then the production numbers will be in the range [0, 9].

Each kind of robot attacks an opponent (or opponents in the case of SpreadBot) following a different strategy. You will override the attack method in the Robot subclasses based on the following information:

- SimpleBot: Attacks a random robot on the opposite team.
- PredatorBot: Attacks the opposite team's robot with the highest health.
- DefenceBot: Attacks the opposite team's robot with the lowest speed.
- SpeedBot: Attacks the opposite team's robot with the lowest attack.
- SpreadBot: Attacks the slowest three robots in the opposite team. If the opposite team has less than three robots, it attacks them all.
- OneBot: Attacks the opposite team's robot with the lowest health.

The attack method receives the Simulation object as a parameter so that you can determine the target robot(s) for the attack. To this end, include the following methods in the Simulation class to be called by the attack method:

- `Robot getRandomTarget(boolean isRedTeam)`: Return a random Robot instance from the red or blue team. If the isRedTeam parameter is true the method should choose the random robot from the red team, otherwise from the blue team.
- `Robot getHighestHealth(boolean isRedTeam)`: Returns the robot with the highest health from the desired team.
- `Robot getLowestHealth(boolean isRedTeam)`: Returns the robot with the lowest health from the desired team.
- `Robot getLowestSpeed(boolean isRedTeam)`: Returns the robot with the lowest speed from the desired team.
- `Robot getLowestAttack(boolean isRedTeam)`: Returns the robot with the lowest attack from the desired team.
- `Robot[] getLowestSpeed3(boolean isRedTeam)`: Returns the Robot array containing the desired team's three robots with the lowest speed values. This method can return 1, 2, or 3 Robot instances based on the remaining robots of the desired team.

After determining the target to attack, the attacking robot’s attack value is subtracted from the target’s health value using the `getHitAndIsDestroyed(double damage)` method. This

method returns true if the target is destroyed; in this case, you must remove the destroyed robot from its team. Include a `void removeRobot(Robot r)` method in the Simulation class to remove the destroyed robots within the overridden attack methods. If the removed robot was going to attack, the order continues with the next robot. You can keep integer indexes for both teams to keep the order of attacking, make sure these indexes do not exceed the team sizes.

Include a menu-driven program in your main method to get the team size from the user, then run the simulation printing out the robots' attacks. At the end, display the winning team and the remaining health of the robots in that team. Note that you should only print three digits after the decimal point for any of the values; you may use `String.format("%.3f", value)` for this. You can add additional methods and variables to make the program functional.

A sample output is given as follows:

```
Team Size: 7

Red team:
X1 Health: 1,475 Attack: 1,954 Speed: 3,945
X6 Health: 1,419 Attack: 1,443 Speed: 3,494
K0 Health: 2,801 Attack: 0,659 Speed: 1,065
P2 Health: 2,604 Attack: 2,101 Speed: 0,999
O3 Health: 0,892 Attack: 4,327 Speed: 0,788
P5 Health: 2,834 Attack: 2,143 Speed: 0,654
O4 Health: 0,794 Attack: 4,340 Speed: 0,545

Blue team:
X13 Health: 1,751 Attack: 1,835 Speed: 3,388
S8 Health: 2,611 Attack: 1,004 Speed: 1,947
S9 Health: 2,162 Attack: 1,615 Speed: 1,668
S11 Health: 2,639 Attack: 1,244 Speed: 1,641
K12 Health: 2,707 Attack: 0,850 Speed: 1,150
P10 Health: 2,378 Attack: 2,311 Speed: 0,660
D7 Health: 4,243 Attack: 0,542 Speed: 0,533

Speed sum of Red: 11,490
Speed sum of Blue: 10,987

Red starts first.

X1 attacks D7
D7 receives 1,954 damage -> remaining health: 2,289
X13 attacks O4
O4 receives 1,835 damage -> remaining health: 0,000
O4 destroyed.
X6 attacks D7
D7 receives 1,443 damage -> remaining health: 0,845
```

S8 attacks X6
X6 receives 1,004 damage -> remaining health: 0,415
K0 attacks following targets:
D7 P10 K12
D7 receives 0,659 damage -> remaining health: 0,186
P10 receives 0,659 damage -> remaining health: 1,719
K12 receives 0,659 damage -> remaining health: 2,048
S9 attacks K0
K0 receives 1,615 damage -> remaining health: 1,186
P2 attacks S11
S11 receives 2,101 damage -> remaining health: 0,538
S11 attacks P2
P2 receives 1,244 damage -> remaining health: 1,359
O3 attacks D7
D7 receives 4,327 damage -> remaining health: 0,000
D7 destroyed.
K12 attacks following targets:
P5 O3 P2
P5 receives 0,850 damage -> remaining health: 1,984
O3 receives 0,850 damage -> remaining health: 0,042
P2 receives 0,850 damage -> remaining health: 0,509
P5 attacks S8
S8 receives 2,143 damage -> remaining health: 0,468
P10 attacks P5
P5 receives 2,311 damage -> remaining health: 0,000
P5 destroyed.
X1 attacks P10
P10 receives 1,954 damage -> remaining health: 0,000
P10 destroyed.
X13 attacks O3
O3 receives 1,835 damage -> remaining health: 0,000
O3 destroyed.
X6 attacks K12
K12 receives 1,443 damage -> remaining health: 0,605
S8 attacks X6
X6 receives 1,004 damage -> remaining health: 0,000
X6 destroyed.
K0 attacks following targets:
K12 S11 S9
K12 receives 0,659 damage -> remaining health: 0,000
K12 destroyed.
S11 receives 0,659 damage -> remaining health: 0,000
S11 destroyed.
S9 receives 0,659 damage -> remaining health: 1,503
S9 attacks X1
X1 receives 1,615 damage -> remaining health: 0,000
X1 destroyed.

```
P2 attacks X13
X13 receives 2,101 damage -> remaining health: 0,000
X13 destroyed.
S8 attacks K0
K0 receives 1,004 damage -> remaining health: 0,182
K0 attacks following targets:
S9 S8
S9 receives 0,659 damage -> remaining health: 0,844
S8 receives 0,659 damage -> remaining health: 0,000
S8 destroyed.
S9 attacks P2
P2 receives 1,615 damage -> remaining health: 0,000
P2 destroyed.
K0 attacks following targets:
S9
S9 receives 0,659 damage -> remaining health: 0,185
S9 attacks K0
K0 receives 1,615 damage -> remaining health: 0,000
K0 destroyed.

Blue team wins, remaining robots:
S9 Health: 0,185 Attack: 1,615 Speed: 1,668
```

Preliminary Submission: You will submit an early version of your solution before the final submission. This version should at least include the following:

- The following classes should be completed: Simulation, Robot, SimpleBot.

You will have time to complete your solution after you submit your preliminary solution. You can consult the TAs and tutors during the lab. Do not forget to make your final submission at the end.

Even if you finish the assignment in the preliminary submission, you should submit for the final submission on Moodle.

Not completing the preliminary submission on time results in a 50% reduction of this assignment's final grade.