COMP2396B Object-oriented Programming and Java

Assignment 1

Due date: 4th March 2022 (Friday), 23:59

Introduction

This assignment tests your understanding of basic inheritance in Java.

You are required to implement three classes to model some robots, which will enter a combat with each other.

There are two special kinds of robot:

- The **PowerBot** can boost itself for a stronger attack.
- The **NinjaBot** can hide itself to dodge an attack.

You are required to implement a **Robot** class, which represents a robot in general, and two classes, **PowerBot** and **NinjaBot**, which are the subclasses of the **Robot** class.

For this assignment, you are required to write JavaDoc for all classes and all methods. A **RobotCombat** class will be provided to simulate a fighting between some robots.

Implementation

The *Robot* class:

Properties

`String name`
: Name of the robot

`int power`
: Power level of the robot

`int lifeIndex`
: Life index of the robot, which indicates the undamaged level of the robot. The robot will be consider broken when this value drops to zero or below.

Methods (all method must be public)

`Robot(String name, int power, int lifeIndex)`
: Constructor, which setup the three properties accordingly

`String getName()`
: Return the name of the robot

`void damage(int amount)`
: Reduce the life index of the robot by the specific amount.
It must print out "*`robot_name`* `takes a damage of` *`amount`*`!`", with *`robot_name`* and *`amount`* replaced by the appropriate values.

`int attack()`

Return the power of the next attack. It always return the value of the level property. It must print out "*robot_name* launches an attack!", with *robot_name* replaced by an appropriate value.

`boolean isBroken()`

Return `true` if its life drops to zero or below.

The *PowerBot* class, a subclass of the *Robot* class:

Methods (all method must be public)

`PowerBot(String name, int power, int lifeIndex)`

Constructor, which setup the three properties accordingly

`void boost()`

Boost itself for the next attack. It must print "*robot_name* boosts itself!", with *robot_name* replaced by an appropriate value.

`int attack()`, an overridden method

If boosted, return twice of the level value if it is boosted. Otherwise return the level value as usual. Reset boosting effect afterwards.

If boosted, it must print "*robot_name* makes a heavy strike!". Otherwise it must print "*robot_name* strikes hard!" (*robot_name* must be replaced by an appropriate value).

The *NinjaBot* class, a subclass of the *Robot* class:

Methods (all method must be public)

`NinjaBot(String name, int power, int lifeIndex)`

Constructor, which setup the three properties accordingly

`void hide()`

Hide itself from the next attack. It must print "*robot_name* hides itself from attacks!", with *robot_name* replaced by the appropriate values.

`void damage(int amount)`, an overridden method

If hidden, it takes no damage at all. Otherwise reduce life with the amount specified as usual. Reset the hiding status afterwards.

If hidden, it must print "*robot_name* hides from the attack!". Otherwise it must print "*robot_name* takes a damage of *amount*!" as usual. (*robot_name* and *amount* must be replaced by an appropriate value.)

You may add additional properties and methods to these three classes.

The main program

A sample main class, *RobotCombat* is given to test your program. Here is the code in the main method of the class:

```
public class RobotCombat {
public static void main(String[] args) {

PowerBot pBot = new PowerBot("Super Droid", 10, 15);
NinjaBot nBot = new NinjaBot("Ninja Sakura", 10, 15);
System.out.println("Now fighting: " + pBot.getName() + " vs. " +
nBot.getName());

int round = 0;
while (!pBot.isBroken() && !nBot.isBroken()) {
    if (round % 2 == 0) {
        int attackAmount = pBot.attack();
        nBot.damage(attackAmount);
        if (round % 3 == 0) {
            nBot.hide();
        }
    } else {
        int attackAmount = nBot.attack();
        pBot.damage(attackAmount);
        if (round % 3 == 1) {
            pBot.boost();
        }
    }
    round++;
}

if (pBot.isBroken())
    System.out.println(nBot.getName() + " wins!—");
else
    System.out.println(pBot.getName() + " wins!—");

}
}
```

Sample Output

(once you've finished programming, running the given *RobotCombat.java* will give you the following output)

```
Now fighting: Super Droid vs. Ninja Sakura
Super Droid strikes hard!
Ninja Sakura takes a damage of 10!
Ninja Sakura hides itself from attacks!
Ninja Sakura launches an attack!
Super Droid takes a damage of 10!
Super Droid boosts itself!
```

```
Super Droid makes a heavy strike!
Ninja Sakura hides from the attack!
Ninja Sakura launches an attack!
Super Droid takes a damage of 10!
Ninja Sakura wins!
```

A Piece of Intelligence:

If doc comments for ~~the~~ JavaDoc ~~is~~ are not complete, you will have at most 15% of your marks deducted.

You will get 0 mark if:

- you submit class files instead of java source files, or
- you submit java source files that cannot be compiled, or
- you submit java source files that are downloaded from the Internet, or
- you submit java source files from your classmates, or
- you submit java source files from friends taken this course last year.

Your program must be compliable and executable with the given **RobotCombat** class. ~~Notice that we may test your program with a different **RobotCombat** class implementation.~~

Submission:

Please submit the following files to Moodle and evaluate. ~~in one compressed file~~

- *Robot.java*
- *PowerBot.java*
- ~~*NinjaBot.java*~~
- ~~*doc.zip, a zip file containing the generated JavaDoc*~~

You must evaluate your program before the assignment due date in order to get marks. The auto-grade system gives you the maximum mark of 85% for the correctness of program output. It tests your program with the **RobotCombat** class and the return value of each class method. Testcases are generally hidden, but you could get hints from the Execution tab to fix bugs if your program fails to pass.

TAs will manually assign the remaining 15% of marks for JavaDoc after the due date. Make sure the doc comments in your submitted source files can generate proper JavaDoc on your computer. You don't need to submit the generated JavaDoc.

Formatted: Underline

Formatted: Justified

Formatted: Font: (Default) Times New Roman, (Asian) Times New Roman, 12 pt, Italic

Formatted: Font: (Default) Times New Roman, (Asian) Times New Roman, 12 pt

Formatted: Left, Indent: Left: 0 cm, First line: 0 cm, Tab stops: 5.37 cm, Centered