OOP & Classes

Lab 02

Overview

- Lab Test
- String Format
- Class & Objects Basics
- Game Simulation Practice

Lab Test

- It will be a closed-book, closed-internet, individually-done test.
- Duration is 30 minutes.
- There will be a total of 3 questions in the lab test.
- We will provide a skeleton code (`.java` file) for each question in the test.
- After finishing up your test, please zip all three `.java` files into one file and name it as your student number.

Lab Test

1. Download the lab test zip file (LabTest01_skeleton.zip) from eTL.
2. Unzip the .zip file.
3. Open IntelliJ IDEA and make a new project.
4. Drag and drop the three .java skeleton files to the src folder in the IDEA project.
5. Fill out the codes.
6. Zip the three completed .java files into one .zip file and name it as your student number (201X-XXXXX.zip).
7. Upload the .zip file to eTL.

String formatting

- Use a format string and String.format method instead of multiple string chunks and + operators.
  - A format string contains format specifiers for variable types, such as “%s” for string, “%d” for `int`/`long`, “%f” for float, etc.

```
String name = "Jack",
    studentID = "2013-12690";
int age = 23;
String str =
    "id: " + studentID + ", name: " + name + ", age: " + age;
String formatString =

    String.format("id: %s %s %d", studentID, age, 23);
```

Format specifier

Format string

`str` and `formatString` have the same value.

String formatting

- To print formatted string to console, use System.out.printf instead of System.out.println().
    - You need to add newline character ‘\n’ at the end of the string format, because printf doesn’t break line.

```
String name = "Jack",
    studentID = "2013-12690";
int age = 23;
System.out.println("id: " + studentID
    + ", name: " + name + ", age: " + age);
System.out.printf("id: %s %s %d\n", studentID, age, 23);
```

Class Basics - Classes and Objects

- All Java programs are written inside something called a “class.”
- Classes are the blueprints of objects.
- Objects are the actual instances of “things.”
- Objects of the same class share similar properties, or attributes.
- Objects of the same class are able to do similar things with methods.

Simple Class Examples - Attributes

Class Definition

```
class Car {
    int carNumber;
    String model;

    Car(int number, String modelName) {
        this.carNumber = number;
        this.model = modelName;
    }
}
```

Main Function

```
Car myCar = new Car(1234, "Sonata");
Car yourCar = new Car(4321, "SantaFe");

System.out.println(myCar.carNumber); // 1234
System.out.println(yourCar.carNumber); // 4321
```

Simple Class Examples - Methods

Class Definition

```
class Car {
    ...
    public void printCarInfo() {
        System.out.println(this.carNumber+" "+this.modelName);
    }
}
```

Main Function

```
Car myCar = new Car(1234, "Sonata");

myCar.printCarInfo(); // 1234 Sonata
```

Constructors - Default Constructors

```
class Car { }

class Car {
    Car() { }
}
```

Same!

```
Car newCar = new Car();
```

Constructors - Doing Something

```
class Car {
    Car() {           Without any parameters
        System.out.println("Car object is created!");
    }

    Car(String message) {           With some parameters
        System.out.println(message);
    }
}
```

```
Car newCar1 = new Car();
Car newCar2 = new Car("I am a new car!");
```

```
Car object is created!
I am a new car!
```

Constructors - Initializing Attributes

Class Definition

```
class Car {
    int carNumber;
    String model;

    Car(int carNumber, String model) {
        this.carNumber = carNumber;
        this.model = model;
        System.out.println("Car initialized.");
    }
}
```

Main Function

```
Car myCar = new Car(1234, "Sonata");
System.out.println(myCar.carNumber, " ", mayCar.model);
```

```
Car initialized.
1234 Sonata
```

Default Attribute Initialization

Class Definition

```
class Car1 {
    int carNumber;
    String model;
}
```

```
class Car2 {
    int carNumber = 9999;
    String model = "Default Model";
}
```

Main Function

```
Car newCar1 = new Car1();
System.out.println(newCar1.carNumber); // 0
System.out.println(newCar1.model); //

Car newCar2 = new Car2();
System.out.println(newCar2.carNumber); // 9999
System.out.println(newCar2.model); // Default Model
```

Methods

```
class Car {
    String location = "Home";
    public void driveToWork() {
        this.location = "Work";
        System.out.println("vroom vroom...");
    }
    public void whereAmI() {
        System.out.println("I am at " + this.location);
    }
}
```

```
Car myCar = new Car();
myCar.whereAmI(); // I am at Home
myCar.driveToWork(); // vroom vroom...
myCar.whereAmI(); // I am at Work
```

Game Simulation

- Write a program that creates two players who fight each other.
- There are three components to this program: Player, Fight, and Main.

Player Class

- A player has a unique userId.
- Each player has a fixed amount of health at the beginning. A player loses when his/her health point reaches zero.
- At each round, a player can either attack or heal.
    - At each attack, a player attacks the opponent with a random attack point in the range of 1 through 5.
    - At each healing, a player can randomly heal his/her health point in the range of 1 through 3.
- A player has a series of attack/heal strategy called tactics.

Player - Attributes/Constructor

```
public class Player {

    String userId;

    int health = 50;
    char[] tactics;

    Player(String userId) {
        this.userId = userId;
        generateRandomTactics();
    }
...
```

Player - Attack/Heal

```
public void attack(Player opponent) {
    opponent.health -= (int)(Math.random() * 5) + 1;
    if (opponent.health < 0) {
        opponent.health = 0;
    }
}

public void heal() {
    health += (int)(Math.random() * 3) + 1;
    if (health > 50) {
        health = 50;
    }
}
```

Player - Helper Methods

```
public char getTactic(int round) {
    return tactics[round];
}

public boolean alive() {
    return health > 0;
}

public void generateRandomTactics() {
    tactics = new char[200];
    for(int i = 0; i < 200; i++) {
        double r = Math.random();
        if (r > 0.3) {
            tactics[i] = 'a';
        } else {
            tactics[i] = 'h';
        }
    }
}
```

Fight Class

- Fight class manages the interactions between the players
- A fight is defined as a session in which two players fight until there is a winner or the time limit is reached.
- A fight keeps track of the rounds.
    - At each round, a fight gathers each player’s tactic (attack/heal) and takes care of what happens to each player’s health points.
- After a round is over, a fight checks if there is a winner or if the time limit was reached.

Fight - Attributes/Constructor

```
public class Fight {

    int timeLimit = 100;
    int currRound = 0;

    Player p1;
    Player p2;

    Fight(Player p1, Player p2) {
        this.p1 = p1;
        this.p2 = p2;
    }
...
```

Fight - Rounds Management

```
...
    public void proceed() {
        System.out.println("Round " + currRound);
        attackHeal();
        currRound++;
    }
...
```

Fight - Rounds Management

```
...
    public void attackHeal() {
        char p1Tactic = p1.getTactic(this.currRound);
        char p2Tactic = p2.getTactic(this.currRound);

        if (p1Tactic == 'a') {
            System.out.println(p1.userId + " attacks " + p2.userId);
            p1.attack(p2);
        } else {
            System.out.println(p1.userId + " heals");
            p1.heal();
        }

        if (p2Tactic == 'a') {
            System.out.println(p2.userId + " attacks " + p1.userId);
            p2.attack(p1);
        } else {
            System.out.println(p2.userId + " heals");
            p2.heal();
        }
    }
...
```

Fight - Helper Methods

```
public boolean isFinished() {
    boolean limitReached = currRound >= timeLimit;
    boolean p1Alive = p1.alive();
    boolean p2Alive = p2.alive();
    return limitReached || !p1Alive || !p2Alive;
}

public Player getWinner() {
    if (p1.health > p2.health) {
        return p1;
    } else {
        return p2;
    }
}

public void printPlayerHealth() {
    System.out.println(p1.userId + " health: " + p1.health);
    System.out.println(p2.userId + " health: " + p2.health);
}
```

Main Class

- Main class is where we actually define the players and the fight.
- We manage the flow of the game in this class.

Main

```
public class Main {

    public static void main(String[] args) {

        Player me = new Player("Do Il");
        Player you = new Player("Hyunwoo");

        Fight testFight = new Fight(me, you);

        while(!testFight.isFinished()) {
            testFight.proceed();
            testFight.printPlayerHealth();
        }
        String winnerId = Fight.getWinner(me, you).userId;
        System.out.println(winnerId + " is the winner!");
    }
}
```