Object-oriented Programming (OOP)

Lab 3

TA : Changmin Jeon, Minji Kim, Hyunwoo Jung,
Hyunseok Oh, Jingyu Lee, Seungwoo Jo

SEOUL NATIONAL UNIVERSITY

Announcement

- You should finish the lab practice and submit your job to eTL before the next lab class starts (Wednesday, 6:30 PM).
- The answer of the practice will be uploaded after the due.

Overview

- Java Basic Review
- String Format
- Class & Objects Basics
- Practice 1, 2 - Java Basics

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, name: %s, age: %d", studentID, age);
```

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 - Doing Something

```
class Car {
    Car() {
        System.out.println("Car object is created!");
 }

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

Without any parameters

With some parameters

```
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

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

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

Main Function

```java
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
```

Practice 1 - X Drawn with Xs

- Write a program that gets integer N (N>=1) as input repeatedly and prints a large X with multiple uppercase “X” and “O” characters.
- Name your source file as XDrawing.java.
- End your program when you get 0 as input.

Practice 1 - X Drawn with Xs

	Console
Input	3
Output	X0X
	0X0
	X0X
Input	4
Output	X00X
	0XX0
	0XX0
	X00X

Practice 2 - Sieve of Eratosthenes

- Write a program that gets integer N (N>=1) as input repeatedly and prints all the prime numbers smaller than N each time.
- Your output prime numbers should be printed in 1 line and sorted in ascending order.
- If there is no prime number smaller than N, print an empty line.
- End your program when you get 0 as input.
- Name your source file as Eratosthenes.java.

Practice 2 - Sieve of Eratosthenes

Console

Input	5
Output	2 3
Input	10
Output	2 3 5 7
Input	2
Output	
Input	15
Output	2 3 5 7 11 13
Input	0
Output	

Submission

- Compress your Practice1.java and Practice2.java file into 20XX-XXXXX_{name}.zip - for example, 2020-12345_KimMinji.zip
- Upload it to eTL - Lab 3 assignment.

Practice - Game Simulation

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

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

```java
public class Main {
    public static void main(String[] args) {
        Player superman = new Player("Superman");
        Player batman = new Player("Batman");

        Fight fight = new Fight(superman, batman);

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

Player Class

- Member variables of Player class:
  - `String name`
  - `char[] actions`
    - The action for each round is either ‘a’ or ‘h’. ‘a’ means “attack”, and ‘h’ means “heal”.
    - If the player attack the opponent, the opponent’s health point is decreased by the corresponding value.
    - If the player heal, his/her own health point increases by the corresponding value.
  - `int[] values`
    - Values corresponding to the actions.
  - `int health`
    - Each player has a fixed amount of health (10) at the beginning. A player loses when his/her health point reaches zero.

Fight Class

- A fight instance manages the interactions between the players.
- A fight instance keeps track of the rounds.
  - At each round, a fight instance runs players' actions starting with `Player p1`.
- A fight ends when one of the players lose his/her all health points, or it reaches the maximum round (The rounds starts from 0 and ends at 9 inclusive).

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 - 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);
}
```