Human-Centered
Computer Systems Lab

Object-oriented Programming (OOP)

Lab 3

TA : Hyuna Seo, Kichang Yang, Minkyung Jeong, Jaeyong Kim

SEOUL NATIONAL UNIVERSITY

Announcement

- You should finish the lab practice and submit your job to eTL before the next lab class starts (Wednesday, 7:00 PM).
- There is NO attendance check for this lab. You only need to submit the practice to get the point.
- The answer of the practice will be uploaded after the due.

Overview

- Recap: Class & Objects Basics
- Problem

Recap: 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.

Recap: Class Attributes

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); // null

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

Recap: Constructors

```
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!
```

Recap: Constructors

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
```

Recap: 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
```

Recap: Static members

Class Definition

```
class Car {
   static int num;
   static int totalMile;
   int mile;
   Car() { num++; }
   void setMile(int mile) {
       this.mile = mile;
       totalMile += mile;
   }
}
```

main Method

```
Car car1 = new Car(),
    car2 = new Car(),
    car3 = new Car();
car1.setMile(20);
car2.setMile(30);
car3.setMile(40);
System.out.println(Car.num);
System.out.println(Car.totalMile);
```

Output

```
3
90
```

Recap: Static members

Class Definition

```
class Car {
  static void whatAmI() {
      System.out.println("I am a car");
  }
}
```

main Method

```
Car.whatAmI()
```

Output

```
I am a car
```

Recap: Static members

Class Definition

```
class Car {
  float fuel;
  static float totalFuel() {
    return fuel;
  }
}
```

main Method

```
Car car1 = new Car();
```

Output: Compilation error

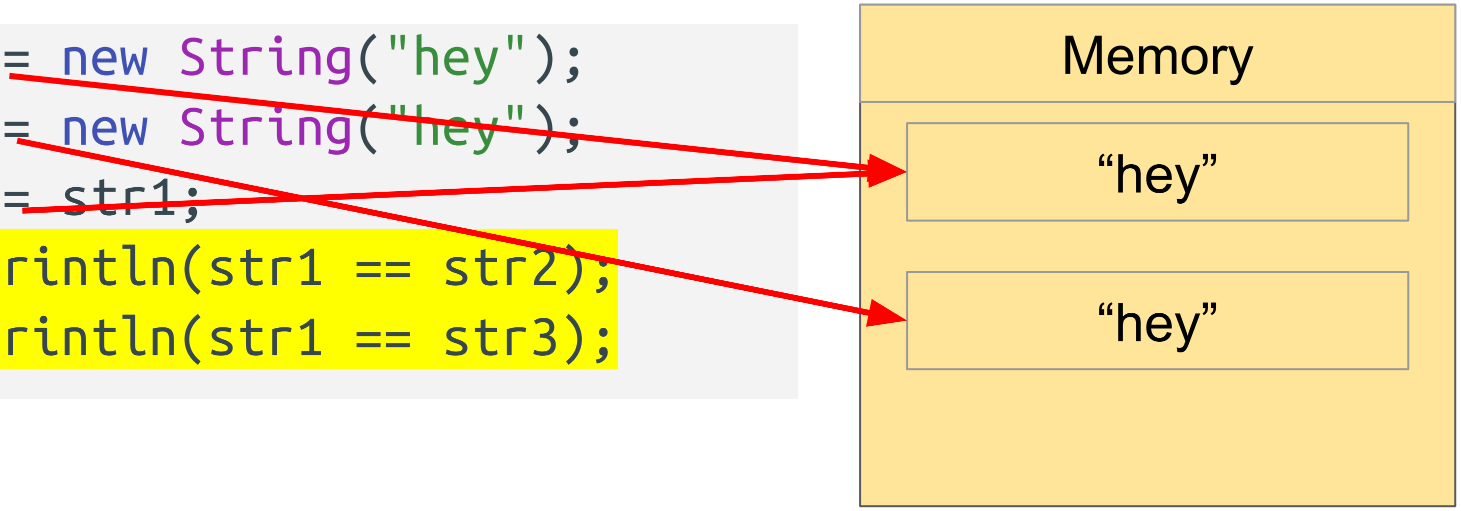
```
java: non-static variable fuel cannot be referenced from a
static context
```

Recap: Object equality

- `==` compares the addresses (not the contents) of the two objects.

Main Function

```
String str1 = new String("hey");
String str2 = new String("hey");
String str3 = str1;
System.out.println(str1 == str2);
System.out.println(str1 == str3);
```



Output

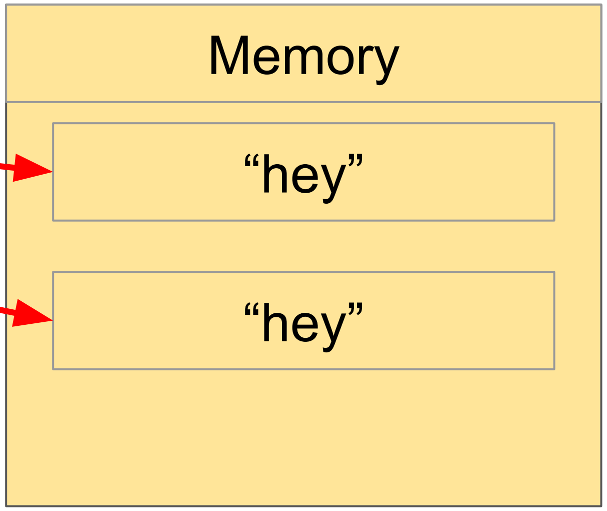
```
false
true
```

Recap: Object equality

- `equals` compares the content of the two objects.

Main Function

```
String str1 = new String("hey");
String str2 = new String("hey");
System.out.println(str1 == str2);
System.out.println(str1.equals(str2));
```



Output

```
false
true
```

Problem - Game Simulation

- Write a program that creates two players who fight each other. At each round, the players take turn attacking/healing. At the end, the program determines the winner.
- There are three classes you need to implement in this program:
    - Player class
    - Fight class
    - Main class

Player Class

- Member variables of Player class:
  - `String userId`
  - `int health`
    - Each player has a fixed amount of health (50) at the beginning. A player loses when his/her health point reaches zero.

Player Class

- Methods of Player class:
  - `public void attack(Player opponent)`
    - Decrease the opponent's health point by a random integer value between 1 (inclusive) and 5 (inclusive). Note that the health point cannot be negative.
    - Hint: Use Math.random()
  - `public void heal()`
    - Increase the player's health point by a random integer value between 1 (inclusive) and 3 (inclusive). Note that the health point cannot exceed the initial health value (50).
  - `public boolean alive()`
    - Return true if the player's health is higher than 0. Otherwise, return false.
  - `public char getTactic()`
    - Decide whether to attack or heal. Attack with a 70% chance and heal with a 30% chance.
      If the player decided to attack, return character 'a'. Otherwise, return 'h'.

Player - Attributes/Constructor

```
public class Player {

    String userId;

    int health = 50;

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

    // TODO: problem1
    ...
```

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 100 inclusive).

Fight Class

- Member variables of Fight class:
  - `int timeLimit`
    - Maximum number of rounds. (Default: 100)
  - `int currRound`
    - Current round. (Initial round: 0)
  - `Player p1, Player p2`
    - Two players that are fighting.

Fight Class

- Methods of Fight class:
  - `public void proceed()`
    - Print current round number in the following format:
      Round [round number]
    - Proceed one round.
      Runs player's actions starting from player 1.
    - Print the health of two players in the following format:
      [player1 userID] health: [player1 health]
      [player2 userID] health: [player2 health]
  - `public boolean isFinished()`
    - Return true if the fight is over. Otherwise, return false.
    - The fight ends when one of the players lose his/her all health points, or it reaches the maximum round.
  - `public Player getWinner()`
    - Return the winner of the fight.
    - The player with more health point wins the fight. If the two players have the same health point, player 2 wins the fight.

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;
    }

     // TODO: problem2
     ...
```

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 Class

- Methods of Main class:
  - `public static void main(String[] args)`
    - Create two players with ID “Superman” and “Batman”.
    - Create Fight instance.
    - Proceed the fight until the fight is over.
    - Print the winner of the fight in the following format:
      [userID] is the winner!

Main Class

```
public class Main {
    public static void main(String[] args) {
        // TODO: problem3
    }
}
```

Submission

- Compress your Main.java, Player.java, Fight.java file into 20XX-XXXXX_{name}.zip - for example, 2020-12345_YangKichang.zip
- Upload it to eTL - Lab 3 assignment.