AP CSA Lecture 4 Sunny

Friday, December 9, 2022 5:20 PM

What we are going to cover today?

1. Quick recap about material in last lecture
2. Talk about last homework, especially "plot" method
3. "this" key word
4. Method overload
5. What is **object reference**
6. Method and reference
7. Preview next homework

1. Recap something about last lecture

2. Plot method

```
public class MovingParticle {
    // The file is originally created by Xuhui Liu (Bison). Please do not distribute it without my permission
    // You will need to implement the MovingParticle class. Everything you need to do is marked with TODO:
    // The class just represents a moving particle in the coordinate system

    // Define instance variable
    private double current_x;
    private double current_y;

    public MovingParticle() {
        // TODO:
        // initialize current coordinate to (0, 0)

    }

    public double get_current_x() {
        // TODO:
        // return current_x
        return 0;
    }

    public double get_current_y() {
        // TODO:
        // return current_y
        return 0;
    }

    public void move_north(double distance) {
        // TODO:
        // move the y coordinate of the particle by distance to north

    }

    public void move_south(double distance) {
        // TODO:

    }

    public void move_west(double distance) {
        // TODO:

    }

    public void move_east(double distance) {
        // TODO:

    }
```

```
    public void shrink_x_half(int n) {
        // TODO:
        // You need to multiply x coordinate by 0.5 for n times

    }

    public void shrink_y_half(int n) {
        // TODO:
        // You need to multiply y coordinate by 0.5 for n times

    }

    public void teleport(double target_x, double target_y) {
        // TODO:
        // move your current x,y to target location

    }

    public void strange_move () {
        // TODO:
        // first take the integer part of your x,y coordinate. If your x,y coordinate are both even, move north by 5
        // If your x coordinate is even, but y coordinate is odd, shrink_x_half 3 times. If your y coordinate is even,
        // but x coordinate is odd, shrink_y_half twice. If your x, y are odd, move south by 5.

    }

    public static void plot(MovingParticle p,  int size) {
        // TODO:
        // Assume the particle p is in 第一象限
        // plot the position of the particle in a coordinate of size: size by size
        // for example, if the particle is in (1,1), size is 5, you should generate the following plot
        /*

        ------
        ------
        ------
        ------
        -#----
        ------

        Basically, you will have a 6 by 6 plot filled with "-" where the particle position is filled with "#"

        */


    }
```

1. Get the current position of the particle p
2. Loop through size by size

```
int x = (int) p.get_current_x();
int y = (int) p.get_current_y();

for (int i = 0; i <= size; i++) {
    for (int j = 0; j <= size; j++) {

    }
}
```

3. If (I, j) == (x, y) you should print a *, otherwise print -

```
int x = (int) p.get_current_x();
int y = (int) p.get_current_y();

for (int i = 0; i <= size; i++) {
    for (int j = 0; j <= size; j++) {
        if (i == x && j == y) {
            System.out.print(s: "*");
        }
        else {
            System.out.print(s: "-");
        }
    }
    System.out.println();
}
```

Don't forget to start a new line after inner loop

4. Suppose we are at (2,3) and want to plot on a plot of size 10

```
MovingParticle p1 = new MovingParticle();
MovingParticle.plot(p1, size: 10);
```

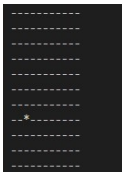
We do so because plot is a static method

```
----------
----------
---*------
----------
----------
----------
----------
----------
----------
----------
----------
```

What is the problem?

We are not having a standard coordinate. Our x coordinate becomes y, and y becomes x since each row is i and each column is j. However, i is 0 in the first row, but we want i to be 0 in the last row. So, you actually want x == j and size - y == i

```
int x = (int) p.get_current_x();
int y = (int) p.get_current_y();

for (int i = 0; i <= size; i++) {
    for (int j = 0; j <= size; j++) {
        if (i == (size - y) && j == x) {
            System.out.print(s: "*");
        }
        else {
            System.out.print(s: "-");
        }
    }
    System.out.println();
}
```



3. **"this" keyword**
   It is used to refer instance variable or method **inside** the class

   For example:

Original bird class

```
class Bird {

    public double weight_public = 2;
    private double weight_private = 2;

    private double weight;

    // basically, you can use both private and public variable anywhere in the class
    public double test_use = weight_private;

    public static boolean living_condition_good_or_not = true;

    // // use constructor to initialize instance variable
    public Bird(double initial_weight) {
        weight = initial_weight;
    }

    // this method allow object to use private variable
    public double get_weight() {
        // see, you can use private variable anywhere inside the class
        return weight;
    }

    public void eat(double amount) {
        weight = weight + amount;
    }

    // private double square(double x) {
    //     // f(x) = x^2
    //     return x*x;
    // }


    public static void change_living_condition() {
        living_condition_good_or_not = !living_condition_good_or_not;
    }
}
```

New bird class

```
class Bird {

    public double weight_public = 2;
    private double weight_private = 2;

    private double weight;

    // basically, you can use both private and public variable anywhere in the class
    public double test_use = weight_private;

    public static boolean living_condition_good_or_not = true;

    // // use constructor to initialize instance variable
    public Bird(double initial_weight) {
        this.weight = initial_weight;
    }

    // this method allow object to use private variable
    public double get_weight() {
        // see, you can use private variable anywhere inside the class
        return this.weight;
    }

    public void eat(double amount) {
        this.weight = this.weight + amount;
    }

    // private double square(double x) {
    //     // f(x) = x^2
    //     return x*x;
    // }


    public static void change_living_condition() {
        living_condition_good_or_not = !living_condition_good_or_not;
    }
}
```

4. **Method Overload**

Method Overload

Method can have the same

Name but different signature

(parameters), but there can't be

Two methods with the same

Name and different return type

Overloaded methods are two or more methods in the same class that have the same name but different parameter lists. For example,

```
public class DoOperations
{
    public int product(int n) { return n * n; }
    public double product(double x) { return x * x; }
    public double product(int x, int y) { return x * y; }
    ...
```

The compiler figures out which method to call by examining the method's *signature*. The signature of a method consists of the method's name and a list of the parameter types. Thus, the signatures of the overloaded `product` methods are

```
product(int)
product(double)
product(int, int)
```

Note that for overloading purposes, the return type of the method is irrelevant. You can't have two methods with identical signatures but different return types. The compiler will complain that the method call is ambiguous.

Having more than one constructor in the same class is an example of overloading. Overloaded constructors provide a choice of ways to initialize objects of the class.

Two method having the same name must have different signature that is to say either have **different types of parameters** or have **different number of parameters**

Suppose you have a method Do_something(). The following overloading is allowed:

1. Do_something(int x)
2. Do_something(int x, int y)
3. Do_something(double y)
4. Do_something(int x, double y)

Let's check a bird example about constructor overloading

You can have two constructors with different signatures

```java
// // use constructor to initialize instance variable

public Bird(double initial_weight) {
    this.weight = initial_weight;
}

public Bird() {
    this.weight = 10;
}
```

5. Object reference

Primitive type and reference type

All of the numerical data types, like `double` and `int`, as well as types `char` and `boolean`, are *primitive* data types. All objects are *reference* data types. The difference lies in the way they are stored.

Consider the statements

```
int num1 = 3;
int num2 = num1;
```

The variables `num1` and `num2` can be thought of as memory slots, labeled `num1` and `num2`, respectively:

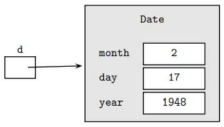
If either of the above variables is now changed, the other is not affected. Each has its own memory slot.

Pointer

Contrast this with the declaration of a reference data type. Recall that an object is created using `new`:

```
Date d = new Date(2, 17, 1948);
```

This declaration creates a reference variable `d` that refers to a `Date` object. The value of `d` is the address in memory of that object:

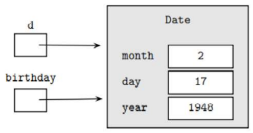


Pointer

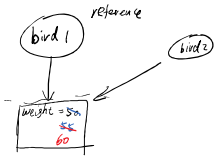
Suppose the following declaration is now made:

```
Date birthday = d;
```

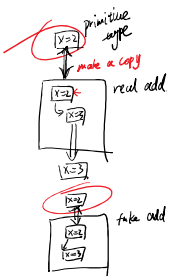
This statement creates the reference variable `birthday`, which contains the same address as `d`:




```java
// the following is about reference
Bird bird1 = new Bird(initial_weight: 50);

// this is null reference
Bird bird2;

// add weight of bird1
bird1.eat(amount: 5);
System.out.println(bird1.get_weight());
// now the weight of bird1 is 55

// let bird2 refer to bird1
bird2 = bird1;

// add weight of bird2
bird2.eat(amount: 5);

// Now the weight of bird1 becomes 60
System.out.println(bird1.get_weight());
```



6. Method and Reference



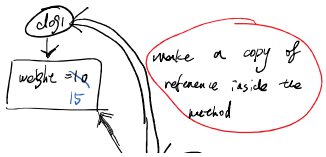
When the method parameter is primitive type

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

    int x = 2;
    fake_add(x);
    // Now guess what is the value of
    System.out.println(x); > 2

    x = real_add(x);      a = realAdd(x);
    // Now guess what is the value of
    System.out.println(x);
```

```java
public static void fake_add(double x) {
    x = x + 1;
}

public static int real_add(int x) {
    x = x + 1;
    return x;
}
```

改变 object 的 instance variable 的值

When method parameter is a reference type and you want to do something to the object



In Dog.java

```java
public class Dog {
    private double weight = 10;
```

In client program:

```java
public static void force_dog_to_eat(Dog d){
    d.eat(amount: 5);
```

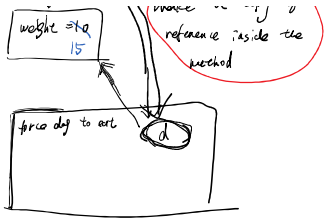
In Dog.java

```
public class Dog {
    private double weight = 10;

    public void eat(double amount) {
        this.weight = this.weight + amount;
    }

    public double get_weight() {
        return this.weight;
    }
}
```

In client program:

```
public static void force_dog_to_eat(Dog d){
    d.eat(amount: 5);
}

public static void main (String args[]) {

    Dog dog1 = new Dog();
    force_dog_to_eat(dog1);
    // Now guess what is the value of weight of dog1
    System.out.println(dog1.get_weight());
}
```

15

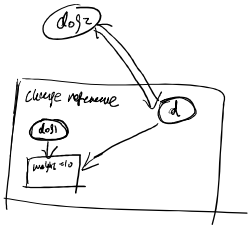


When method parameter is a reference type and you want to change the reference

```
public static void change_reference(Dog d) {
    Dog dog1 = new Dog();
    d = dog1;
}
```

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

    Dog dog2 = null;
    change_reference(dog2);
    // Now guess what is the value of dog2
    System.out.println(dog2.get_weight());
}
```

null



In conclusion, you can pass an object to the method parameter and change the instance variable of that object, but you cannot change the reference of that object.

7. Preview next homework

You need to implement a Crab and Shark interaction program. The Shark class is already implemented for you. You only need to code for the Crab.java and Main.java. Please clearly follow the instruction in the java file and address all the "TODO" mark.

```
public class Main {
    Run | Debug
    public static void main(String args[]) {
        // TODO: create a shark and a crab and do the following sequence of operation
        // 1. crab attack the shark, and shark attack the crab
        // 2. shark fierce attack the crab and the crab attack the shark by lianhuantui
        // 3. shark heals for 10 point and crab heals for 10 points
        // 4. print out the remaining hp for both crab and shark

        Shark shark = new Shark();
        Crab crab = new Crab();


    }
}
```

What shark.java looks like

```
public class Shark {
    private double hp;
    private double offense;
    private double defense;


    public Shark() {
        // Initialize instance variable. Initialize hp to 100, offense to be 30 and defense to be 2
        this.hp = 100;
        this.offense = 30;
        this.defense = 2;
    }
    public Shark(double hp, double offense, double defense) {
        this.hp = hp;
        this.offense = offense;
        this.defense = defense;
    }

    public double view_hp() {
        // return the shark's hp
        return this.hp;
    }

    public double view_offense() {
        return this.offense;
    }

    public void under_attack(Crab crab, double crab_offense) {
        // deduct the hp of the shark by the following formula hp_change = max(opponent offense - shark defense, 0)
        // that is deducting (crab's offense - shark defense) if it is greater than 0, otherwise deduct 0
        if (crab_offense - this.defense >= 0) {
            this.hp = this.hp - (crab_offense - this.defense);
        }
    }

    public void heal(double hp_change) {
        // add the shark's hp by hp_change
        this.hp = this.hp + hp_change;
    }

    public void attack(Crab crab) {
        // attack the given crab. Notice that you need to use crab.under_attack()
        crab.under_attack(this, this.offense);
    }

    public void fierce_attack(Crab crab) {
        // this is similar to attack except that your offense becomes 1.5 times original offense
        crab.under_attack(this, this.offense*1.5);
    }
```

What you need to implement

```
public class Crab {
    private double hp;
    private double offense;
    private double defense;

    private int num_qianzi;
    private int num_legs;

    public Crab() {
        // TODO: initialize instance variable
        // Initialize hp to 100, offense to 12, defense to 15, num_qianzi to 2, and num_legs to 8

    }

    public double view_hp() {
        // TODO: return current hp
        return 0;
    }

    public double view_offense() {
        return this.offense;
    }

    public int view_qianzi() {
        // TODO: return the number of qianzi of current crab
        return 0;
    }

    public int view_legs() {
        // TODO: return the number of legs of current crab
        return 0;
    }


    public void under_attack(Shark shark, double shark_offense) {
        // TODO: deduct the hp of the crab by the following formula hp_change = max(opponent offense - carb defense, 0)
        // that is deducting opponent offense - shark defense if it is greater than 0, otherwise deduct 0
        // if the hp_change is greater than 20, the crab will loss a leg

    }

    public void heal(double hp_change) {
        // TODO: add the crab's hp by hp_change
    }

    public void attack(Shark shark) {
        // TODO: attack the given shark. Notice that you need to use shark.under_attack()
    }
```

Mutator method : 改变 instance variable

Accessor method : 读取 instance variable