COMP 1006/1406 B1 MIDTERM ASSIGNMENT 1

Jacob Chiu

101271070

Java Classes

A class can define a new data type, can be a program can be a collection of functions and data, or it can be some combination of all of these.

Declaration

things that are static belong to the class and not objects.

what can we say about static attributes?

1. they exist even if an object of the class does not (think of Math)
2. public attributes can be accessed/modified by any object or class in the program
3. usually not a good idea unless they are constants (final)
4. private attributes are only accessible within the class (information hiding!)
5. static attributes are stored in the static area in the memory

Public

public here is a top-level access modifier

1. it specifies the access level of the class (which other classes can access it)
2. a public class can be accessed by any other class that can see it
3. you can restrict access to a class by using different access modifiers (private, protected or no modifier at all (default))

Private

Secret

Modifiers

Static

static here is a non-access modifier

1. static declares main to be a class method instead of being an instance method (which is the default)
2. this means that the main method belongs to the HelloWorld class itself and not individual HelloWorld objects
3. it allows the main method to be called without needing an instance of the HelloWorld class.

Parameters

Byte, Int, Long, Double, String, Bool, array

Return type

Byte, Int, Long, Double, String, Bool, array, void

Body

Constructors

we can think of constructors as creation or initialization methods, but

1. constructors must have the exact same name as the class
2. constructors have no return value (not even void)
3. constructors can only be called with the new operator (and one other time that we will see soon)
4. constructors are NOT methods (although they are similar)

This Keyword

Java keyword this

1. a reference to the current object
2. used in constructors and instance methods
3. this is needed here because attributes name and id are not in scope (the input parameters name and id are in scope)

Constructor Chaining

calling a constructor from within another constructor of the same class is called constructor chaining.

Methods

Methods of an object typically act on (use and modify) the state of that object.

Getters and Setters

Value Types

Byte, Int, Long, Double, String, Bool, array

Inheritance

Extends key word

@overrides

Scope

Abstraction

things that are final cannot be changed once they are defined.

Ex.

```
public final int x = 3;
public final String str = "cat";
public final Student s = new Student("dog", 4);
public final int[] numbers = {1,3,5,7,9};
```

ONLY the value in the variable is held constant.

Java Memory

- Stack
  - Variable assignment
  - Stack overflow error
  - Activation Records

    an activation record (or stack frame) is a data structure associated with method and constructor calls

    1. when a method (or constructor) is called, an activation record is created and added (pushed) to the top of the memory stack
    2. think of a stack of plates... we add (push) a plate to the top and we remove (pop) a plate from the top. We never access the middle or bottom of the stack
    3. a stack follows the LIFO principle (last-in-first-out)
    4. stores all local variables of the method/constructor
    5. method parameters
    6. variables declared inside the method
    7. stores other information needed by JVM to run your program (like which line of code to return to in calling method)
    8. when method ends, its activation record is removed (popped) from the stack and all local variables are lost to the program.
- Heap
  - Variable address
  - Garbage Collection
- Static area

Encapsulation

Encapsulation refers to two ideas

1. classes and objects have both state and behavior
2. the internal details of the data are hidden.

We'll look at the second idea more now

- often called **information hiding**
- related to idea of **separation of concerns**
  - actual code and how you use the code are independent
- access to data is restricted
  - **getter** or **accessor** methods allows us to see the data
  - **setter** or **mutator** methods allows us to change the data
  - not all data will be visible and not all data will be allowed to be modified
- why would we want to do this?

```
//  A
public class Person {
    public String name     = "";
    public String favPet   = "";
    public String[] petLis;

    Person(String name, String favPet, String[] pets) {
        this.name = name;
        this.favPet = favPet;
        this.petLis = pets;
    }
    public boolean SuperHappy() {
        int count = 0;
        for (int i = 0; i < this.petLis.length; i++) {
            if (this.petLis[i] == this.favPet) {
                count += 1;
```

```
            }
        }
        if (count >= (this.petLis.length + 1) / 2){
            return true;
        }
        else{
            return false;
        }
    }

    public static String[][] countPet(String pet, Person[] people) {
        String[][] countPetLis = new String[people.length][];
        for (int i = 0; i < people.length; i++) {
            int count = 0;
            for (int k = 0; k < people[i].petLis.length; i++) {
                if (people[i].petLis[k] == pet) {
                    count += 1;
                }
            }
            countPetLis[i][0] = people[i].name;
            countPetLis[i][1] = Integer.toString(count);
            count = 0;
        }
        return countPetLis;
    }
    public static void main(String[] args){

    }
}


public class QQ {
    private final int q;
    public QQ(int q) {
        this.q = q;
    }
    public int get_q() {
        return this.q;
    }
}
public class WW extends QQ {
    public String w;
    WW(int q, String w){
        super(q);
        this.w = w;
```

```
    }
}
```