FUNDAMENTALS

CS62

DATA STRUCTURES AND ADVANCED PROGRAMMING

5: Inheritance

Alexandra Papoutsaki
she/her/hers

Lecture 5: Inheritance

- Inheritance

```
package registrar;
class PomonaStudent {
    private String name;
    private String email;
    private int id;
    private String major;
    private static int studentCounter;

    protected PomonaStudent(String name, String email, int id){
        this.name = name;
        this.email = email;
        this.id = id;
        studentCounter++;
        major = "Undeclared";
    }
    //protected setters getters
    protected int getMaxCredits(){
        return 4;
    }
    public String toString(){
        return “Pomona Student Info - Name: " + name + "\nemail: " + email + "\nid: " + id + "\n";
    }
}
```

Students across different years have some unique characteristics

- First-year students take ID1
- Fourth-year students write a thesis
- Second-year students and above can take 6 credits
- Transfer students take 1 PE class instead of 2, etc.
- But they still are Pomona students so the basic information we would need about them doesn't change.

Inheritance

- When you want to create a new class and there is already a class that includes some of the code that you want, you can derive your new class from the existing class. In doing this, you can reuse the variables and methods of the existing class without having to write (and debug!) them yourself.
- A class that is derived from another is called a subclass or child class.
- The class from which the subclass is derived is called a superclass or parent class.
- Single inheritance: A class can only extend ONE AND ONLY one parent class.
- Multilevel inheritance: A class can extend a class which extends a class etc.

Inheritance

- The subclass inherits all the `public` and `protected` members.
    - Not the `private` ones, although it can access them with appropriate getters and setters.
- The inherited fields can be used directly, just like any other fields.
- You can write a new instance method in the subclass that has the same signature as the one in the superclass, thus overriding it.

All classes inherit class `Object`

- Directly if they do not extend any other class, or indirectly as descendants.
- `Object` class has built-in methods that are inherited.
- `public boolean equals (Object other)`
  - Default behavior uses == returns true only if this and other are located in same memory location.
  - Works fine for primitives but not objects. We would need to override it (more later).
- `public String toString()`
  - Returns string representation of object – default is hexadecimal hash of memory location.
  - Does NOT print the string.
  - Typically needs to be overridden to be useful.
- `public int hashCode()`
  - Unique identifier defined so that if `a.equals(b)` then `a, b` have same hash code (more later).

`super` keyword

- Refers to the direct parent of the subclass.
- `super.instanceMethod()`: for overridden methods.
- `super(args)`: to call the constructor of the super class. First line in constructor of subclass.

```
package registrar;

class FirstYearPomonaStudent extends PomonaStudent{

    private String id1;
    private static int firstYearCounter;

    protected FirstYearPomonaStudent(String name, String email, int id, String id1){
        super(name, email, id);
        this.id1 = id1;
        firstYearCounter++;
    }
   //getters and setters

  public String toString(){
        return super.toString() + “First-Year Student Attending ID1: " + id1;
    }
}
```

```
package registrar;


class SecondYearPomonaStudent extends PomonaStudent{

    private static int secondYearCounter;

    protected SecondYearPomonaStudent(String name, String email, int id){

        super(name, email, id);

        secondYearCounter++;

    }


    protected int getMaxCredits(){

        return 6;

    }


    public String toString(){

        return super.toString() + “Second-Year Student can Take: " + getMaxCredits() +" credits";

    }

}
```

```
package registrar;


class FourthYearPomonaStudent extends PomonaStudent{

    private String thesisTitle;

    private static int fourthYearCounter;

    protected FourthYearPomonaStudent(String name, String email, int id, String thesisTitle){

        super(name, email, id);

        this.thesisTitle = thesisTitle;

    }

    //getters and setters

   protected int getMaxCredits(){

        return 6;

    }

    public String toString(){

        return super.toString() + "Fourth-Year Student Writing Thesis on: " + thesisTitle;

    }

}
```

PRACTICE TIME - Worksheet

- While extending your animal rescue application to include a `Cat` class you discover that cats and dogs have a lot of common characteristics the shelter cares about (let's say `name`, `age`, `daysInRescue`) and that some of that information could be saved in a parent class `Animal` instead that both `Dog` and `Cat` would extend.
- From your research, you know that people who search to adopt dogs care about their breed while people who search to adopt cats care about their fur length.
- Write three classes, Animal, `Dog`, `Cat`, with the appropriate instance variables, constructors, counters, and toString methods.
- Put the classes in a package and choose the right access modifier for your fields and methods.

Overriding methods

```
FirstYearPomonaStudent s1 = new
FirstYearPomonaStudent("daniel", "daniel@pom.edu", 1, "War and
Peace");

System.out.println(s1);

}
```

- Will print

```
Pomona Student Info - Name: daniel

email: daniel@pomona.edu

id: 1
```

Polymorphism

```
FirstYearPomonaStudent student1 = new FirstYearPomonaStudent("daniel",
"daniel@pomona.edu", 1, "War and Peace");

SecondYearPomonaStudent student2 = new SecondYearPomonaStudent("archita",
"archita@pomona.edu", 3);

FourthYearPomonaStudent student3 = new FourthYearPomonaStudent("antonis",
"antonis@pomona.edu", 6, "Savoir Vivre Around the World");

PomonaStudent[] students = new PomonaStudent[3];

students[0] = student1;

students[1] = student2;

students[2] = student3;

for(PomonaStudent student: students){

   System.out.println(student); //appropriate overriden toString method

   //student.getID1(); //would not work; not a method of the super class

}
```

Polymorphism

- `ParentClass obj = new ChildClass();` E.g.,

```
PomonaStudent student7 = new FirstYearPomonaStudent("alex",
"alex@pomona.edu", 1, "Humans through the eyes of technology”);

System.out.println(student7.getMaxCredits());
```

- Will print 4

```
 student7 = new SecondYearPomonaStudent(student7.getName(),
student7.getEmail(), student7.getId());
```

- Will print 6

Hiding vs overriding

- Overriding: For instance methods this is determined at runtime.
- One form of polymorphism (dynamic).
  - Static polymorphism happens when we overload methods.
- Hiding: For variables (instance+static) and methods (static) that appear in both super and subclass, this is determined at compile-time. Be careful!
- You will get a compile-time error if you attempt to change an instance method in the superclass to a static method in the subclass and vice-versa.

Example: Animal

```
public class Animal {
    public int legs = 2;
    public static String species = "Animal";
    public static void testStaticMethod() {
        System.out.println("The static method in Animal");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Animal");
    }
}
```

Example: Cat

```
public class Cat extends Animal {
    public int legs = 4;
    public static String species = "Cat";
    public static void testStaticMethod() {
        System.out.println("The static method in Cat");
    }
    public void testInstanceMethod() {
        System.out.println("The instance method in Cat");
    }
}
```

Hiding vs overriding

```
public static void main(String[] args) {
    Cat myCat = new Cat();
    myCat.testStaticMethod(); //invoking a hidden method
    myCat.testInstanceMethod(); //invoking an overridden method
    System.out.println(myCat.legs); //accessing a hidden field
    System.out.println(myCat.species); //accessing a hidden field
}
```

- Output:

```
The static method in Cat
The instance method in Cat
4
Cat
```

WHAT YOU WERE EXPECTING, RIGHT?

Hiding vs overriding

```
public static void main(String[] args) {
    Animal yourCat = new Cat();
    yourCat.testStaticMethod(); //invoking a hidden method
    yourCat.testInstanceMethod(); //invoking an overridden method
    System.out.println(yourCat.legs); //accessing a hidden field
    System.out.println(yourCat.species); //accessing a hidden field
}
```

- Output:

```
The static method in Animal
The instance method in Cat
2
Animal
```

!!!

PRACTICE TIME

```
public class ClassA {
    public void methodOne(int i) {
    }
    public void methodTwo(int i) {
    }
    public static void methodThree(int i) {
    }
    public static void methodFour(int i) {
    }
}

public class ClassB extends ClassA {
    public static void methodOne(int i) {
    }
    public void methodTwo(int i) {
    }
    public void methodThree(int i) {
    }
    public static void methodFour(int i) {
    }
}
```

1. Which method *overrides* a method in the superclass?

2. Which method *hides* a method in the superclass?

3. What do the other methods do?

ANSWER

1. `methodTwo.`

2. `methodFour.`

3. They cause compile-time errors.
   `methodOne: “This static method cannot hide the instance method from ClassA”.`
   `methodThree: “This instance method cannot override the static method from ClassA”.`

`final` keyword

- Variable: only assigned once in its declaration or in constructor – its value cannot be changed after initialization.
    - E.g., `static final int PI = 3.14;`
- Method: cannot be overridden by subclass.
- Class: cannot be extended.

Lecture 5: Inheritance

- Inheritance

Readings:

- Inheritance: https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

Code

- Lecture 5 code

Worksheet

- Lecture 5 worksheet