Java Classes and Information Hiding

Readings: Chapter 2

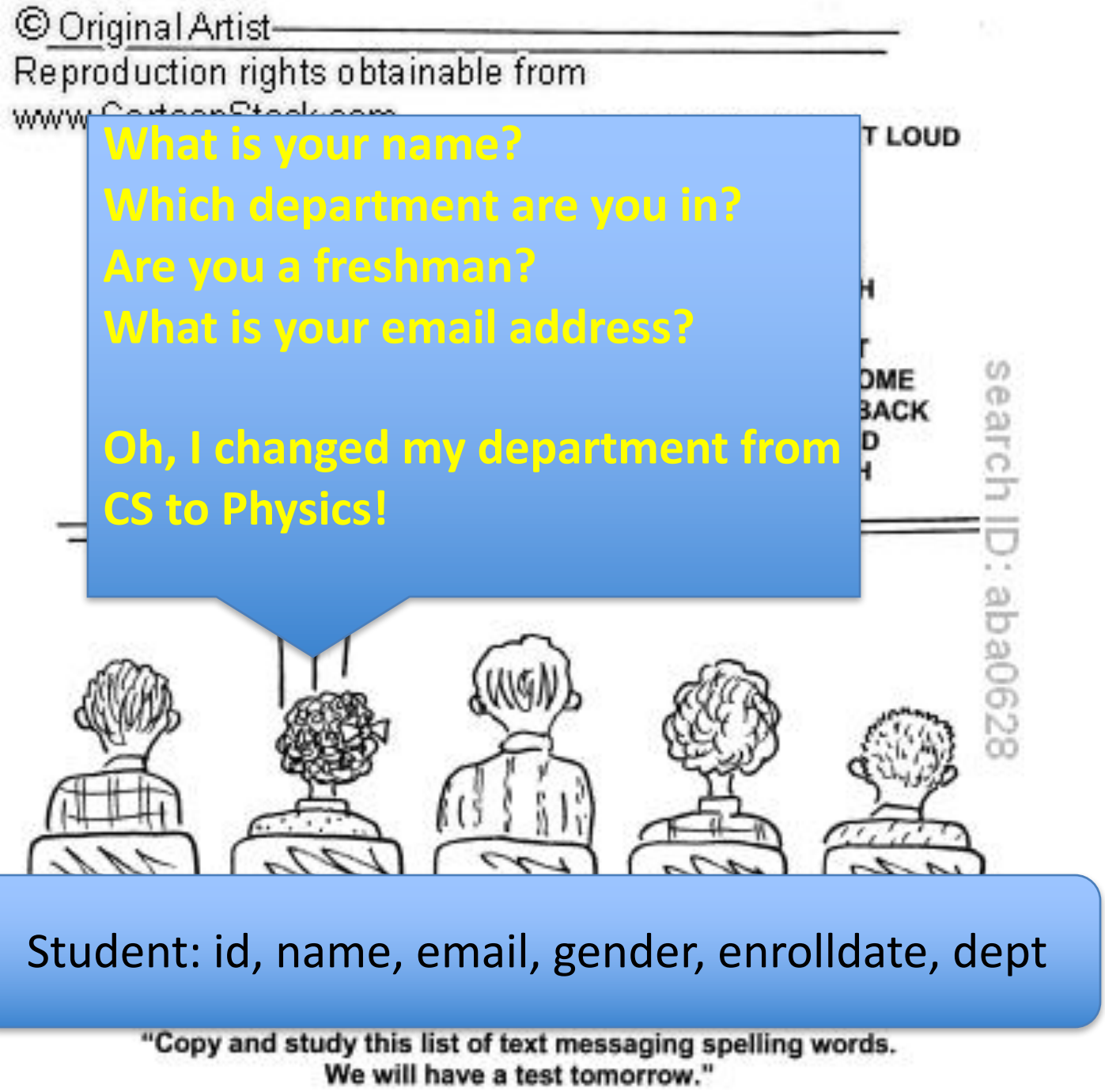
Outline

- OOP
- Define a class
- Use a class

Object-oriented Programming (OOP)

- OOP is a relatively new approach to programming which supports
  - the creation of new data types
  - operations to manipulate those types (methods)
- Eight primitive data types in Java
  - byte (8-bit), short (16-bit), int(32-bit), long(64-bit)
  - float (32-bit), double(64-bit)
  - boolean (represent 1bit,size?), char (16-bit Unicode character)
  - http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
- Java
  - class: create objects and methods

Student: id, name, email, gender, enrolldate, dept



Student: id, name, email, gender, enrolldate, dept

Outline

- OOP
- Define a class
- Use a class

Object and Members

- Class: define a new type of object (or data type).
- Members:
  - Variables (information, type)
  - Constructors: designed to provide initial values to the class's data
  - Methods to answer the questions

Instance variable

```
public class Student {
        private int id;
        private String name;
        private int age;
        private String email;
        private boolean gender;
        private Date enrolldate;
        private String dept;

        ...
}
```

Constructor

- Purpose: designed to provide initial values to the class's data
- Syntax:
  - Default value (for any number is zero)
  - Name
  - No return value
- Example: constructor for a student
- No-argument constructor
- Notes: duplicate

Constructor for a student

```
public Student(int _id, String _name){
        id = _id;
        name = _name;
}

public Student(int _age, String _name){
        age = _age;
        name = _name;
}
public Student(int _id, String _name, int _age){
        id = _id;
        name = _name;
        age = _age;
}
```

Constructor for a student

```
public Student(int _id, String _name){
        id = _id;
        name = _name;
}

public Student(int _age, String _name){
        age = _age;
        name = _name;
}
public Student(int _id, String _name, int _age){
        id = _id;
        name = _name;
        age = _age;
}
```

OK!

Duplicate!

OK!

Copy constructor

```
public class Location implements Cloneable{
    private double x; // The x coordinate of this Location
    private double y; // The y coordinate of this Location
    public Location(Object o){
        if(o==null) return;
        if(o instanceof Location){
            Location l = (Location) o;
            x = l.x;
            y = l.y;
        }else{
            return;
         }
    }
…
}
```

Why use Object as parameter type?

```
public class LocationSubclass extends Location{
    private String color = "";
    ...
    public LocationSubclass(Object o){
        super(o);
        color = "black";
    }
    public String toString(){return (super.toString()+",color="+color);}
    public static void main(String[] args) {
        LocationSubclass loc1 = new LocationSubclass();
        Location loc2 = new Location(loc1);

        //This will not work if not using Object type
        LocationSubclass loc3 = new LocationSubclass(loc2);
    }
}
```

Method

- Accessor method
  - Make no changes to the object's instance variables
- Modification method
- Terminology
  - call a method = activate a method

Example

```
public class Location {
    private double x; // The x coordinate of this Location   private
    double y;
    public Location(double _x, double _y){
        x = _x;
        y = _y;
    }
    public double getX( ){…}
    public void setX(double px){…}
```

Example (cont.)

```
public double getX( ){
    return x;
}
public void setX(double px){
    x = px;
}
```

Access modifiers

- Public
- Private
- Default access (or package access)
  - When no modifies are associated to member definition
  - The member can be accessed only by other classes in the same package

Package

- Why do we need packages?
- Name a package
- Use a package: import
- Java Class Libraries (JCL)
  - ArrayList, Vector, Hashtable, and HashMap (Appendix D)
- Unnamed package (http://docs.oracle.com/javase/specs/jls/se7/html/jls-7.html)
  - Classes defined in unnamed packages cannot be imported

Outline

- OOP
- Define a class
- Use a class

Use a class

- Create new objects and refer to them by names

  Student stu1 = new Student();

- Import class
  - import cs272.Student;
  - Import edu.nmsu.cs272.Student;
- Activate a method
  - stu1.setName(“Sarah”);

```
public class LocationTest {
    public static void main(String[] args) {
        Location loc1 = new Location(1.0,1.0);
        System.out.println("loc1 x: "+loc1.getX());
        loc3.setX(3.0);
        System.out.println("loc1 x: "+loc1.getX());
    }
}
```

What is the output?

```
public class LocationTest {
    public static void main(String[] args) {
        Location loc1 = new Location(1.0,1.0);
        System.out.println("loc1 x: "+loc1.getX());
        loc3.setX(3.0);
        System.out.println("loc1 x: "+loc1.getX());
    }
}
```

What is the output?

loc1 x: 1.0

loc1 x: 3.0

Java object type

- A primitive variable is not an object: byte, short, int, long, char, float, double, boolean
- Everything else is an object
  - A String, a Location, an array, etc.

```
public class LocationTest {
    public static void main(String[] args) {
        Location loc1 = new Location(1.0,1.0);
        System.out.println("loc1: "+loc1);
    }
}
```

What is the output?

```
public class LocationTest {
    public static void main(String[] args) {
        Location loc1 = new Location(1.0,1.0);
        System.out.println("loc1: "+loc1);
    }
}
```

What is the output?

loc1: ch2class.Location@7852e922

toString()

```
/**
  * Generate a String representation of this Location.
  * @param - none
  * @return
  *   a String representation of this Location
  **/
  public String toString( )
  {
     return "(x=" + x + "  y=" + y + ")";
  }
```

Parameters, equal, clone

- Parameters (reference variable, null reference)
- “=“ vs. assignment
- Clone
- “==“ vs. equal
- Copy constructor

Reference variable

- A program with several objects

  Student stu_cs = new Student (‘CS’);

  Student stu_ee = new Student (‘EE’);

- Each object has its own copies of the instance variables
- Reference variable: used to refer to objects

Reference variable

- Difference between a reference variable (used by Java for all classes) and an ordinary variable (for primitive data types such as int, char)
  - Null
  - Assignment

Null reference

- Null reference
  - A reference variable does not refer to anything. The value of the variable is called null.
  - Good practice: when a program finishes using an object, set the reference variable to null.
- Null pointer exception

Reference variable assignment

- Ordinary variables a and b (int)

  a = b;

- What does it happen?

- Reference variables stu1 and stu2 (Student)

  stu1 = stu2

- What does it happen? (Illustration)

Reference variable assignment

- Ordinary variables a and b (int)

  a = b;

- What does it happen?

- Reference variables stu1 and stu2 (Student)

  stu1 = stu2

- What does it happen? (Illustration)

Make stu1 refer to the same object that stu2 is already referring to.

Assignment …

- Reference variable assignment
  - stu1 = stu2
- Create two separate objects?

```
public class Location implements Cloneable{
...
public Location clone( )  {  // Clone a Location object.
    Location answer;
    try     {
        answer = (Location) super.clone( );
        answer.x = x;
        answer.y = y;     }
    catch (CloneNotSupportedException e){
        System.out.println(e.getMessage());
        throw new RuntimeException ("This class does not
implement Cloneable.");
    }
     return answer;
}
```

Equality test

- stu1==stu2??

- If both are null …
- If both are not null
  - Same object?
  - Different object with the same content?

Equality test

- stu1==stu2

- If both are null, the above expression is true
- If both are not null
  - Same object, true
  - Different object with the same content, false

equals

- Test whether two objects have the same value
- Different from “==” operator

equals

```
Location loc1 = new Location(1.0,1.0);
Location loc2 = new Location(2.0,2.0);
Location loc3 = loc1;
Location loc4 = new Location(1.0,1.0);
Location loc5 = loc1.clone();
Location loc6;

System.out.println("loc1==loc5? " + (loc1==loc5));
System.out.println("loc1.equals(loc5)? " + loc1.equals(loc5));
System.out.println("loc5.equals(null)? " + loc5.equals(null));
```

Check Location.java and LocationTest.java

equals

```
Location loc1 = new Location(1.0,1.0);
Location loc2 = new Location(2.0,2.0);
Location loc3 = loc1;
Location loc4 = new Location(1.0,1.0);
Location loc5 = loc1.clone();
Location loc6;

System.out.println("loc1==loc5? " + (loc1==loc5));
System.out.println("loc1.equals(loc5)? " + loc1.equals(loc5));
System.out.println("loc5.equals(null)? " + loc5.equals(null));
```

false

true

false

Writing equals method

```
public boolean equals(Object obj)
{
    if (obj instanceof Location)
    {
      Location candidate = (Location) obj;
      return (candidate.x == x) && (candidate.y == y);
    }
    else
      return false;
}
```

Object as parameter type for equals()?

```
public class LocationSubclass extends Location{
    private String color = "";
    ...
    public boolean equals(LocationSubclass obj){
        System.out.println("calling equals 1...");
        …
    }
    public boolean equals(Object obj){
        System.out.println("calling equals 2...");
        …
    }

    //main method, next slide
}
```

Assume that in class Location, the method equals() is not defined.

Object as parameter type for equals()?

```
public class LocationSubclass extends Location{
    public static void main(String[] args) {
        LocationSubclass loc1 = new LocationSubclass();
        Location loc2 = new Location(loc1);
        LocationSubclass loc3 = new LocationSubclass(loc2);
        LocationSubclass loc4 = new LocationSubclass(loc1);

        System.out.println("loc1==loc2? "+(loc1==loc2));
        System.out.println("loc1.equals(loc2)? "+(loc1.equals(loc2)));
        System.out.println("loc2.equals(loc1)? "+(loc2.equals(loc1)));
        System.out.println("loc1.equals(loc4)? "+(loc1.equals(loc4)));
    }
}
```

Output?

Object as parameter type for equals()?

```
public class LocationSubclass extends Location{
    public static void main(String[] args) {
        LocationSubclass loc1 = new LocationSubclass();
        Location loc2 = new Location(loc1);
        LocationSubclass loc3 = new LocationSubclass(loc2);
        LocationSubclass loc4 = new LocationSubclass(loc1);

        System.out.println("loc1.equals(loc2)? "+(loc1.equals(loc2)));
        System.out.println("loc2.equals(loc1)? "+(loc2.equals(loc1)));
        System.out.println("loc1.equals(loc4)? "+(loc1.equals(loc4)));
    }
}
```

calling equals 2...loc1.equals(loc2)? false

loc2.equals(loc1)? false

calling equals 1...loc1.equals(loc4)? true

Object parameter

public static double distance(Location p1, Location p2){…}

Location distance (p,s);

When a parameter is an object …

The parameter is initialized to refer to the same object that the actual argument refers to.

Return an object type

- Reference variable of the local method

- Midpoint

static

```
/**
  * Compute the distance between two Locations.
  * @param p1  the first Location
  * @param p2 the second Location
  * @return the distance between p1 and p2
  * @note
  *   The answer is Double.POSITIVE_INFINITY if the distance
  *   calculation overflows. The answer is Double.NaN if either
  *   Location is null.
  **/
  public static double distance(Location p1, Location p2)
  {
      …
  }
```

static

```
/**
  * Compute the distance between two Locations.
  * @param p1  the first Location
  * @param p2 the second Location
  * @return the distance between p1 and p2
  * @note
  *   The answer is Double.POSITIVE_INFINITY if the distance
  *   calculation overflows. The answer is Double.NaN if either
  *   Location is null.
  **/
  public static double distance(Location
  {
      …
  }
```

Static: method is not activated by any object

Location.distance(); ✔

p1.distance();✕

What you know about Objects

- You know how to write a new class type, and place the new class in a package.
- You know how to import the class into a program that uses class type.
- You know how to activate methods.
- But you still need to learn how to write the implementations of a class's methods.

Java: Constant

- Double.NaN: not a number
- Infinity
  - Double.POSITIVE_INFINITY
  - Double.NEGATIVE_INFINITY

Summary

- Define a class
- Use a class
  - Reference variable
  - Parameter and return values
    - Location Demonstration
- == vs. equals
- = vs. clone
- toString

Reference

- Example of declaring a class – page 69
- Example of clone method – page 69 (S.S.)
- Example of equals method – page 70
- Example of get methods – page 70
- Example of static method – page 71