COMS W1114 - Java Lab

Lab 10
Wednesday, April 7, 2004
&
Thursday, April 8, 2004

-1-

Note

•Plagiarism policy

*One word: don't. All homeworks and exams in this course are intended to be done **by yourself**, and with the help of the textbook, teaching assistants, the instructor, and the webboard. You're allowed to discuss problems with classmates, but only in general terms, and you must specifically avoid discussing any solutions.*
Moreover, you'd be amazed how easy it is to detect plagiarism or cheating in both written and programming assignments. Cheaters don't spend tremendous amounts of time masking their copy, because that defeats the purpose and it would be simpler to do the homework themselves. Invariably, therefore, they get caught. An infraction is a zero on the assignment or exam and a referral to your academic dean. .

if you think you could have ended up at the top of this list, contact janak by FRIDAY and explain yourself.

-2-

Some Definitions

- `this` **identifier**
  - every object can access all of its members by name. The *full name* of the identifier is `this.name`
  - The use of `this` is only needed when there is ambiguity over variable names in a particular scope

- **overloading a method** - where you provide different versions of a method, but keep the same name. The parameter list (input) for the two methods must be distinct so that there is no confusion about which is being called:

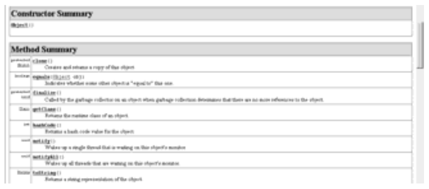
```
public void withdraw(String balance, int double){
   //do stuff here
}
public void withdraw(String balance, String double){
   withdraw(balance, Double.parseDouble(double));
}
public void withdraw(String balance){
   withdraw(balance, 60.0);
}
```

- **overriding methods** – a subclass can decide to supply its own version of a method already supplied.
  - All of the classes we create are a subclass of the Object Class in Java

-3-

Object Class Methods

- equals
  - The equals method for class Object implements the most discriminating possible equivalence relation on objects; that is, for any non-null reference values x and y, this method returns true if and only if x and y refer to the same object (x == y has the value true).
- toString
  - Returns a string representation of the object. In general, the toString method returns a string that "textually represents" this object. The result should be a concise but informative representation that is easy for a person to read. It is recommended that all subclasses override this method. The toString method for class Object returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@', and the unsigned hexadecimal representation of the hash code of the object. In other words, this method returns a string equal to the value of getClass().getName() + '@' + Integer.toHexString(hashCode())



Constructor Summary	
Object()	

Method Summary	
protected Object	clone() Creates and returns a copy of this object.
boolean	equals(Object obj) Indicates whether some other object is "equal to" this one.
protected void	finalize() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class	getClass() Returns the runtime class of an object.
int	hashCode() Returns a hash code value for the object.
void	notify() Wakes up a single thread that is waiting on this object's monitor.
void	notifyAll() Wakes up all threads that are waiting on this object's monitor.
String	toString() Returns a string representation of the object.

-4-

More Definitions

- `instanceof`
  - The instanceof operator tests whether its first operand is an instance of its second. boolean val = op1 instanceof op2 op1 must be the name of an object and op2 must be the name of a class. **An object is considered to be an instance of a class if that object directly or indirectly descends from that class**.

-5-

Program from last lab…

```
public class Plotter {
		final Point origin = new Point(0,0); // a constant

		public static void main(String[] args) {
				Point p1=new Point(3,5);
				Point p2=new Point();

				// try to access vars directly - error
				//p2.x = 3;
				//p2.y = 6;

				System.out.println("p1 x="+ p1.getX()+ " y="+ p1.getY());
				//use toString implicitly instead
				System.out.println("p1 "+p1);
				System.out.println("p2 "+p2);
				System.out.println("Distance between p1 and p2: "+p1.distance(p2));

				//generate a midpoint
				Point mp12=p1.midpoint(p2);
				System.out.println("Midpoint between p1 and p2: "+ mp12); // note overidden toString

				//rotate the midpoint
				mp12.rotate90();
				System.out.println("mp12 after rotate90"+ mp12 );

				//are the midpoint and p1 equal now?
				System.out.println("p1 == mp12? "+p1.equals(mp12));

				//etc....


		}
}
```

3 points created in this Plotter program, each of which have their own ‘copy’ of the member (global) variables

p1
x=3, y=5

p2
x=0, y=0

mp12
x=2, y=2

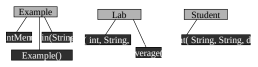
-6-

Questions

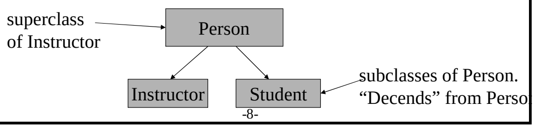
- how could we use instanceof operator here?
  - mp12 instanceof Point
  - mp12 instanceof Object

Inheritance

- **An object is considered to be an instance of a class if that object directly or indirectly descends from that class**.
- descent?!
- so far, we have created classes with no hierarchy between them:
  - remember:



- But is it useful to create a hierarchy when classes are related:



Inheritance Example

```
class Person {
     protected String name;  // a protected variable can be accessed ONLY
     protected int age;      //  from this class and the classes descend from this class…

     public Person(String n, int age){
        name =n;
        this.age = age;
     }
     public String toString(){
        return name;
     }
}

class Student extends Person {  //since student extends Person, student inherits all the properties that Person had…
    private  int studentID;
    private double GPA;
    public Student(String n, int a, int id, double gpa){
          super(n,a);    // we are calling the super-class's constructor MUST BE THE FIRST LINE OF CODE. this is called
                               //superconstructing

          studentID=id;
          GPA = gpa;
    }
    public String toStrint(){
          return "Student: " + name;  //name refers to the PROTECTED Person variable
    }
}
```

Some questions

- Review of modifiers
  - `public`
  - `protected`
  - `private`
- how could we use instanceof operator here?

```
Person p = new Person(“Maryam”, 23);
– p instanceof Person ?
– p instanceof Student ?

Student s = new Student(“Jack”, 19,
  223344, 3.7);
– s instanceof Person ?
– s instanceof Student ?
```

-10-

Casting examples using inheritance properties

- any object of a superclass can be assigned to a subclass with a cast!
  - `Person p = new Person(“Maryam”, 23);`
  - `Student s = (Student) p;    // the id and gpa of student s are currently null!!`
- any object of a subclass can be assigned to an object of its superclass!
  - `Student s = new Student(“Brit”, 22, 6655, 2.7);`
  - `Person p = s;`

-11-

Static

- a static variable
  - a member (global) variable which exists only ONCE even though there may be multiple objects created.
  - good for when you want to collect information about an entire class (vs. a single object) such as
    - static int students_created;
    - Student.students_created; ← access it from the class, not the object!
- a static method
  - not part of the a specific object, part of the general class
  - main method must be static. no other methods should be declared static
  - you can never access an object’s main() from another object.

-12-