COMP 110
More about inheritance

Darrell Bethea
June 7, 2011
Questions?
Today in COMP 110

- Some review
- More inheritance
The difference between *return* and printing

- Returning a value from a method does NOT print it out to the screen
  - The returned value is given to the caller of the method
  - It is up to the method caller to do something with this value
The difference between `return` and printing

```java
public class NumberUtils {
    private static Random rnd = new Random();
    public static int roll20SidedDie() {
        return rnd.nextInt(20) + 1;
    }
}

public class Tester {
    public static void main(String[] args) {
        NumberUtils.roll20SidedDie(); // what does this do?
    }
}
```

- Generates a random die roll, **but** it ignores the value returned from the `roll20SidedDie` method.
- Have we printed anything to the screen?
public class NumberUtils
{
    private static Random rnd = new Random();
    public static int roll20SidedDie()
    {
        return rnd.nextInt(20) + 1;
    }
}

class Tester
{
    public static void main(String[] args)
    {
        int num = NumberUtils.roll20SidedDie(); // what does this do?
    }
}

- Puts the random number returned by roll20SidedDie into the variable num
- Have we printed anything to the screen?
The difference between return and printing

```java
public class NumberUtils {
    private static Random rnd = new Random();
    public static int roll20SidedDie() {
        return rnd.nextInt(20) + 1;
    }
}

public class Tester {
    public static void main(String[] args) {
        int num = NumberUtils.roll20SidedDie(); // what does this do?
        System.out.println(num);
    }
}
```

- Puts the random number returned by `roll20SidedDie` into the variable `num`
- Then prints the value of `num` to the screen
Review: Inheritance

- What is a base class?
- What is a derived class?
We have discussed before how classes of objects can have relationships.
Put these classes into an inheritance hierarchy

- Crocodile
- Human
- Reptile
- Animal
- Mammal
- Whale

Diagram:

```
Animal
  /    \
 Reptile Mammal
   /   /   \
 Crocodile Human Whale
```
This inheritance relationship is known as an *is-a* relationship

- A Doctoral student *is* a Grad student
- A Grad student *is* a Student
- A Student *is* a Person

- Is a Person a Student?
  - Not necessarily!
Review: Inheritance

- Define a general class
- Later, define specialized classes based on the general class
- These specialized classes inherit properties from the general class
public class Person {
    private String name;
    public Person() {
        name = "No name yet";
    }
    public void setName(String newName) {
        name = newName;
    }
    public String getName() {
        return name;
    }
}
Review: How does this work in Java?

```java
public class Student extends Person {
    private int id;
    public Student()
    {
        super();
        id = 0;
    }
    public Student(String stdName, int idNumber)
    {
        setName(stdName);
        setID(idNumber);
    }
    public void setID(int idNumber)
    {
        id = idNumber;
    }
    public int getID()
    {
        return id;
    }
}

Person
- name
+ setName(String newName) : void
+ getName() : String

Student
- id
+ setID(int idNumber) : void
+ getID() : int
```
Override methods

- What if the class Person had a method called printInfo?

```java
public class Person {
    // a bunch of other stuff
    // ...
    public void printInfo() {
        System.out.println(name);
    }
}
```
What if the class Student also had a method called printInfo?

```java
public class Student extends Person {
    // a bunch of other stuff
    // ...
    public void printInfo() {
        System.out.println("Name: " + getName());
        System.out.println("ID: " + getID());
    }
}
```
Both Person and Student have a printInfo() method

Student std = new Student("John Smith", 37183);
std.printInfo(); // calls Student’s printInfo method,
                // not Person’s

Output would be:
Name: John Smith
ID: 37183
You often want derived classes to perform custom behavior

For example, drawing shapes

```java
public class Shape {
    public void draw(Graphics g) {
    }
}
```

```java
public class Rectangle extends Shape {
    public void draw(Graphics g) {
        g.drawRect(...arguments...);
    }
}
```

```java
public class Circle extends Shape {
    public void draw(Graphics g) {
        g.drawOval(...arguments...);
    }
}
```
Type compatibilities

- Given this inheritance hierarchy...

```
Person
  Athlete
    HighJumper
    ExtremeAthlete
      Skydiver
      XGamesSkater
```
Person p = new Person();
  Yes!

Is this code legal?
Is this code legal?

- HighJumper h = new HighJumper();
  - Yes!
Is this code legal?

- Person p = new Athlete();
  - Yes! An Athlete is a Person, so this is okay
Skydiver s = new Person();

No! A Person is not necessarily a Skydiver, so this is illegal
Is this code legal?

- Athlete ath = new Athlete();
  XGamesSkater xgs = ath;
  - No! An Athlete is not necessarily an XGamesSkater, so this is illegal

Diagram:
- Person
  - Athlete
    - HighJumper
    - ExtremeAthlete
      - Skydive
      - XGamesSkater
What’s going on?

- We are creeping up to the idea of polymorphism
  - Enables the substitution of one object for another as long as the objects have the same interface
  - More details later
The class Object

- Every class in Java is derived from the class Object
  - Every class in Java is an Object

```
Object
  Animal
    Reptile
      Crocodile
    Mammal
      Human
      Whale
  Person
    Student
    Employee
```