Lecture 18: Polymorphism (Part II)

Building Java Programs: A Back to Basics Approach

by Stuart Reges and Marty Stepp

Polymorphism

- **polymorphism**:

1) Ability for the same code to be used with different types of objects and behave differently with each.(Part I)
   - `System.out.println` can print any type of object.
     - Each one displays in its own way on the console.

1) Ability for a method to take on many forms. (Part II)

Overriding

- If a child class has the same method as the parent class, the method of the child class **overrides** the method of the parent class.

```
public class ParentClass{
  public void method1(int a){…}
}
public class ChildClass extends ParentClass{
      public void method1(int a){…}
   }
```

Overriding

```
public class Tester{
public static void main(String[] args)
{
     ParentClass a=new ParentClass();
     ParentClass b=new ChildClass();

     a.method1();//calls method1 of ParentClass
     b.method1();//calls method1 of ChildClass
}}
```

Method overriding is also known as **run-time polymorphism** or **dynamic binding**. Java selects the correct method1 at run-time.

Overloading

- A class can have **many forms** of the same method. The methods are distinguished by:

1. Number of parameters
2. Type of the parameters
3. Order of the parameters

Method overloading is also known as **compile-time polymorphism** or **static binding**. Java selects the correct method at compile-time.

Number of Parameters

Methods with the same name can be distinguished by the number of parameters.

```
public class Overload{

public void method1(int c)
{…}

public void method1(int c, double d)
{…}

}
```

Type of Parameters

Methods with the same name can be distinguished by the type of the parameters.

```
public class Overload{

public void method1(int c)
{…}

public void method1(double c)
{…}

}
```

Sequence of Parameters

Methods with the same name can be distinguished by the order of the parameters.

```
public class Overload{

public void method1(int c, double d)
{…}

public void method1(double d, int c)
{…}

}
```

Invalid Overloading

```
Case 1:
public void method1(int c, double d)
{…}

public void method1(int e, double f)
{…}
```

Compile error. Same number, data types and sequence. Methods cannot be overloaded with just different variable names.

```
method1(3,4.1);
```

Invalid Overloading

```
Case 2:
public void method1(int c, double d)
{…}

public boolean method1(int e, double f)
{…}
```

Compile error. Same number, data types and sequence. Even though the return type is different, this is not valid.

```
method1(3,4.1);
```

Ambiguous Call

```
public static void method1(int a, double b)
{…}
public static void method1(double a, int b)
{…}
public static void main(String[] args)
{
    method1(3,4.0);
    method1(3.3,4);
    method1(3,4);
    //error, ambiguous call
}
```

Compile-time vs Runtime

- An error is a **compile-time error** if it happens when the program compiles.
- An error is a **runtime error** if it happens when the program runs.
- A runtime error compiles without errors.

Compile-time vs Runtime

```
Employee Sean=new Secretary();
Sean.takeDictation("hi");
//compile-time error, no such method in Employee

Sean.fileLegalBriefs();
// compile-time error, no such method in Employee

Sean.getHours();
//ok
```

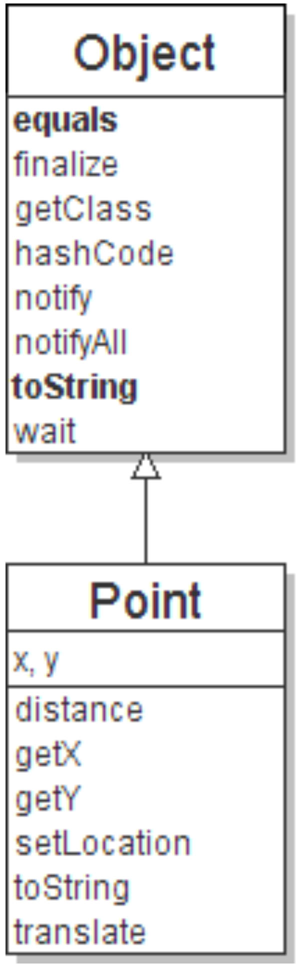
Compile-time vs Runtime

```
((LegalSecretary) Sean).sue();
//compile-time error,sue() isn't in LegalSecretary

((LegalSecretary) Sean).fileLegalBriefs();
//runtime error, cast too far down the tree
//the program compiles without errors.

((Lawyer) Sean).sue();
//runtime error, horizontal casting not allowed;
//the program compiles without errors.
```

The Cosmic SuperClass `Object`

- All types of objects have a superclass named `Object`.
  - Every class implicitly extends `Object`

- The `Object` class defines several methods:
  - `public String toString()`
    Returns a text representation of the object, often so that it can be printed.
  - `public boolean equals(Object other)`
    Compare the object to any other for equality. Returns `true` if the objects have equal state.



`Object` variables

- You can store any object in a variable of type `Object`.

```
Object o1 = new Point(5, -3);
Object o2 = "hello there";
Object o3 = new Scanner(System.in);
```

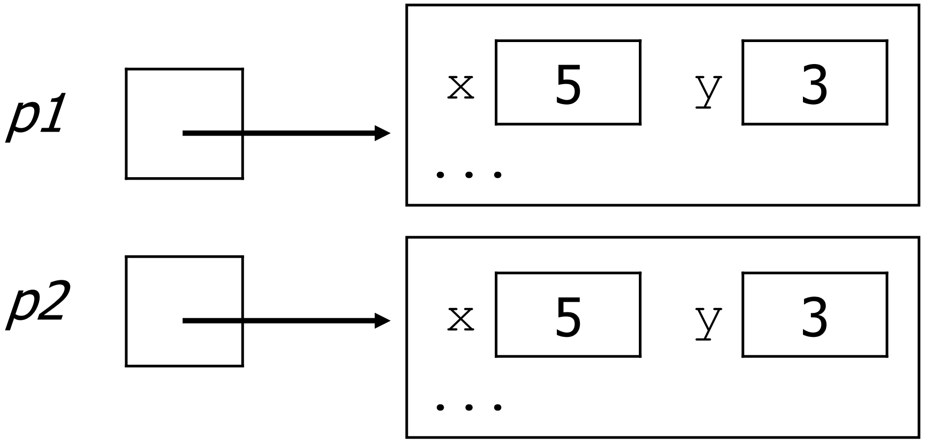
- An `Object` variable only knows how to do general things.

```
String s = o1.toString();       // ok
int len = o2.length();       // compile-time error
String line = o3.nextLine(); // compile-time error
```

Recall: comparing objects

- The `==` operator does not work well with objects.
  - `==` compares references to objects, not their state.
  - It only produces `true` when you compare an object to itself.

```
Point p1 = new Point(5, 3);
Point p2 = new Point(5, 3);
if (p1 == p2) {    // false
    System.out.println("equal");
}
```



The `equals` method

- The `equals` method compares the state of objects.

```
if (str1.equals(str2)) {
    System.out.println("the strings are equal");
}
```

- But if you write a class, its `equals` method behaves like `==`

```
if (p1.equals(p2)) {   // false :-(
    System.out.println("equal");
}
```

  - This is the behavior we inherit from class `Object`.
  - Java doesn't understand how to compare `Point`s by default.

`equals` method

- We can change this behavior by writing an `equals` method.
  - Ours will *override* the default behavior from class `Object`.
  - The method should compare the state of the two objects and return `true` if they have the same x/y position.

```
public boolean equals(Object o) {
    Point other=(Point) o;
    if (x == other.x && y == other.y) {
        return true;
    } else {
        return false;
    }
}
```

`NOTE: Flawed implementation. What if o is NOT a Point?`

The `instanceof` keyword (NOT ON AP Exam)

```
if (variable instanceof type) {
    statement(s);
}
```

- Asks if a variable refers to an object of a given type.
  - Used as a `boolean` test.

```
String s = "hello";
Point p = new Point();
```

expression	**result**
`s instanceof Point`	`false`
`s instanceof String`	`true`
`p instanceof Point`	`true`
`p instanceof String`	`false`
`p instanceof Object`	`true`
`s instanceof Object`	`true`
`null instanceof String`	`false`
`null instanceof Object`	`false`

Final equals method

```
// Returns whether o refers to a Point object with
// the same (x, y) coordinates as this Point.
public boolean equals(Object o) {
    if (o instanceof Point) {
        // o is a Point; cast and compare it
        Point other = (Point) o;
        return x == other.x && y == other.y;
    } else {
        // o is not a Point; cannot be equal
        return false;
    }
}
```

Point Class

```
public class Point{
  int x;
  int y;
  @Override
  //overriding toString of Object class.
  public String toString(){
    return "("+x+","+y+")";}
  @Override //overriding equals of Object class.
  public boolean equals(Object o) {
    if (o instanceof Point) {
        // o is a Point; cast and compare it
        Point other = (Point) o;
        return x == other.x && y == other.y;
    } else {
        // o is not a Point; cannot be equal
        return false;
  }
}}
```

Point Tester

```
public class PointTester{
  public static void main(String[] args){
    Point p1=new Point(2,3);
    Point p2=new Point(2,3);
    System.out.println(p1);  //(2,3), calls toString()
    System.out.println(p1.toString()); //(2,3) explicit call
    //comparing addresses of objects
    if(p1==p2) //false, different objects, different addresses
    {
    …
    }
    //comparing x-y coordinates of Point objects
    if(p1.equals(p2)) //true
    {
    …
    }
}}
```

Lab 1

- Complete the lab on polymorphism posted on Classroom. Download the files polymorphismLab, Circle, Rectangle and Shape. Follow the direction given by the comments.