CSE 143
Lecture 23

Polymorphism; the Object class

read 9.2 - 9.3

slides created by Marty Stepp and Ethan Apter
http://www.cs.washington.edu/143/
Polymorphism

• **polymorphism**: Ability for the same code to be used with different types of objects and behave differently with each.

  – `System.out.println` can print any type of object.
    • Each one displays in its own way on the console.

  – A `Scanner` can read data from any kind of `InputStream`.

  – Every kind of `OutputStream` can write data, though they might write this to different kinds of sources.
Coding with polymorphism

• A variable of type \( T \) can refer to an object of any subclass of \( T \).

\[
\text{Employee } ed = \text{new Lawyer();}
\text{Object otto = new Secretary();}
\]

– You can call any methods from Employee on ed.
– You can \textit{not} call any methods specific to Lawyer (e.g. sue).

• When a method is called on \( ed \), it behaves as a Lawyer.

\[
\text{System.out.println(}ed\text{.getSalary());} \quad // \quad 50000.0
\text{System.out.println(}ed\text{.getVacationForm());} \quad // \quad \text{pink}
\]
Polymorphism/parameters

- You can pass any subtype of a parameter's type.

```java
public class EmployeeMain {
    public static void main(String[] args) {
        Lawyer lisa = new Lawyer();
        Secretary steve = new Secretary();
        printInfo(lisa);
        printInfo(steve);
    }

    public static void printInfo(Employee empl) {
        System.out.println("salary = " + empl.getSalary());
        System.out.println("days = " + empl.getVacationDays());
        System.out.println("form = " + empl.getVacationForm());
    }
}

OUTPUT:
salary = 50000.0    salary = 50000.0
vacation days = 21  vacation days = 10
vacation form = pink vacation form = yellow
```
Coding with polymorphism

• We can use polymorphism with classes like `OutputStream`.
  – Recall methods common to all `OutputStreams`:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>write(int b)</code></td>
<td>writes a byte</td>
</tr>
<tr>
<td><code>close()</code></td>
<td>stops writing (also flushes)</td>
</tr>
<tr>
<td><code>flush()</code></td>
<td>forces any writes in buffers to be written</td>
</tr>
</tbody>
</table>

– Recall part of the inheritance hierarchy for `OutputStream`:

```
OutputStream
   ↓
FilterOutputStream
   ↓
PrintStream
   ↓
FileOutputStream
   ↓
```
Streams and polymorphism

- A variable of type T can refer to an object of any subclass of T.
  
  ```java
  OutputStream out = new PrintStream(new File("foo.txt"));
  OutputStream out2 = new FileOutputStream("foo.txt");
  ```

  - You can call any methods from OutputStream on out.
  - You can not call methods specific to PrintStream (println).
    - But how would we call those methods on out if we wanted to?

- When out runs a method, it behaves as a PrintStream.
  
  ```java
  out.write(0);    // writes a 0 byte to foo.txt
  out.close();    // closes the stream to foo.txt
  ```
Polymorphism examples

• You can use the object's extra functionality by casting.

```java
OutputStream out = new PrintStream(new File("foo.txt"));
out.write(0); // ok
out.println("hello"); // compiler error
((PrintStream) out).println("hello"); // ok
out.close(); // ok
```

• You can't cast an object into something that it is not. Such code might compile, but it will crash at runtime.

```java
OutputStream out2 = new FileOutputStream("foo.txt");
```
Polymorphism mystery

• 4-5 classes with inheritance relationships are shown.

• A client program calls methods on objects of each class.
  – Some questions involve type-casting.
  – Some lines of code are illegal and produce errors.

• You must read the code and determine its output or errors.
  – For output, you must be precise
  – For errors, you need only say that an error occurred (not identify what kind of error occurred)

• We always place such a question on our final exams!
Steps to solving polymorphism mystery problems:

1. Look at the variable type. (If there is a cast, look at the casted variable type.) If the variable type does not have the requested method the compiler will report an error.

2. If there was a cast, make sure the casted variable type is compatible with the object type (i.e. ensure the object type is a subclass of the variable type). If they are not compatible, a runtime error (ClassCastException) will occur.

3. Execute the method in question, behaving like the object type. (The variable type and casted variable type no longer matter.)
Exercise

• Assume that the following classes have been declared:

```java
public class Snow {
    public void method2() {
        System.out.println("Snow 2");
    }

    public void method3() {
        System.out.println("Snow 3");
    }
}

public class Rain extends Snow {
    public void method1() {
        System.out.println("Rain 1");
    }

    public void method2() {
        System.out.println("Rain 2");
    }
}
```
public class Sleet extends Snow {
    public void method2() {
        System.out.println("Sleet 2");
        super.method2();
        method3();
    }

    public void method3() {
        System.out.println("Sleet 3");
    }
}

public class Fog extends Sleet {
    public void method1() {
        System.out.println("Fog 1");
    }

    public void method3() {
        System.out.println("Fog 3");
    }
}
What happens when the following examples are executed?

• Example 1:

```java
Snow var1 = new Sleet();
var1.method2();
```

• Example 2:

```java
Snow var2 = new Rain();
var2.method1();
```

• Example 3:

```java
Snow var3 = new Rain();
((Sleet) var3).method3();
```
Technique 1: diagram

- Diagram the classes from top (superclass) to bottom.
## Technique 2: table

<table>
<thead>
<tr>
<th>method</th>
<th>Snow</th>
<th>Rain</th>
<th>Sleet</th>
<th>Fog</th>
</tr>
</thead>
<tbody>
<tr>
<td>method1</td>
<td></td>
<td>Rain 1</td>
<td></td>
<td>Fog 1</td>
</tr>
<tr>
<td>method2</td>
<td>Snow 2</td>
<td>Rain 2</td>
<td>Sleet 2</td>
<td>Sleet 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>method3()</td>
<td>Snow 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>method3()</td>
</tr>
<tr>
<td>method3</td>
<td>Snow 3</td>
<td>Snow 3</td>
<td>Sleet 3</td>
<td>Fog 3</td>
</tr>
</tbody>
</table>

*Italic* - inherited behavior  
*Bold* - dynamic method call
Example 1

• Example:

```java
Snow var1 = new Sleet();
var1.method2();
```

• Output:

```
Sleet 2
Snow 2
Sleet 3
```
Example 2

- Example:
  ```java
  Snow var2 = new Rain();
  var2.method1();
  ```

- Output:
  None!
  There is an error, because Snow does not have a method1.
Example 3

• Example:

    Snow var3 = new Rain();
    ((Sleet) var3).method2();

• Output:

    None!
    There is an error because a Rain is not a Sleet.
The Object class

read 9.3
Class 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`.

```java
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.

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

- You can write methods that accept an `Object` parameter.

```java
public void checkForNull(Object o) {
    if (o == null) {
        throw new IllegalArgumentException();
    }
}
```
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.

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

Point \( p_1 \) = new Point(5, 3);
Point \( p_2 \) = new Point(5, 3);
if (\( p_1 == p_2 \)) {
    // false
    System.out.println("equal");
}
• The `equals` method compares the state of objects.

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

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

```java
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 `Points` by default.
• 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.

• A flawed implementation:

```java
public boolean equals(Point other) {
    if (x == other.x && y == other.y) {
        return true;
    } else {
        return false;
    }
}
```
• The body can be shortened to the following:

```java
// boolean zen
return x == other.x && y == other.y;
```

• It should be legal to compare a `Point` to any object (not just other `Points`):

```java
// this should be allowed
Point p = new Point(7, 2);
if (p.equals("hello")) {  // false
    ...

- equals should always return false if a non-Point is passed.
```
public boolean equals(Object name) {
    statement(s) that return a boolean value;
}

- The parameter to equals must be of type Object.
- Object is a general type that can match any object.
- Having an Object parameter means any object can be passed.
  - If we don't know what type it is, how can we compare it?
Another flawed version

• Another flawed `equals` implementation:

```java
public boolean equals(Object o) {
    return x == o.x && y == o.y;
}
```

• It does not compile:

`Point.java:36: cannot find symbol
symbol : variable x
location: class java.lang.Object
return x == o.x && y == o.y;`

```java
^```

The compiler is saying,
"o could be any object. Not every object has an x field."
Type-casting objects

• Solution: *Type-cast* the object parameter to a `Point`.

  ```java
  public boolean equals(Object o) {
      Point other = (Point) o;
      return x == other.x && y == other.y;
  }
  ```

• Casting objects is different than casting primitives.
  – Really casting an `Object` reference into a `Point` reference.
  – Doesn't actually change the object that was passed.
  – Tells the compiler to *assume* that `o` refers to a `Point` object.
• Client code:

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

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

Diagram:

- `p1` and `p2` both have `x = 5` and `y = 3`.
- `p1` and `p2` are compared using the `equals` method.
Comparing different types

```java
Point p = new Point(7, 2);
if (p.equals("hello")) {  // should be false
    ...
}
```

− Currently our method crashes on the above code:

```
Exception in thread "main"
java.lang.ClassCastException: java.lang.String
    at Point.equals(Point.java:25)
    at PointMain.main(PointMain.java:25)
```

− The culprit is the line with the type-cast:

```java
public boolean equals(Object o) {
    Point other = (Point) o;
```
The `instanceof` keyword

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

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

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

<table>
<thead>
<tr>
<th>expression</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>s  instanceof  Point</td>
<td>false</td>
</tr>
<tr>
<td>s  instanceof  String</td>
<td>true</td>
</tr>
<tr>
<td>p  instanceof  Point</td>
<td>true</td>
</tr>
<tr>
<td>p  instanceof  String</td>
<td>false</td>
</tr>
<tr>
<td>p  instanceof  Object</td>
<td>true</td>
</tr>
<tr>
<td>s  instanceof  Object</td>
<td>true</td>
</tr>
<tr>
<td>null instanceof  String</td>
<td>false</td>
</tr>
<tr>
<td>null instanceof  Object</td>
<td>false</td>
</tr>
</tbody>
</table>
// 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;
    }
}