Polymorphism

Coding with polymorphism

- A variable of type $T$ can hold an object of any subclass of $T$.
  
  ```java
  Employee ed = new Lawyer();
  ```
  
  - You can call any methods from the Employee class on ed.

- When a method is called on ed, it behaves as a Lawyer.
  
  ```java
  System.out.println(ed.getSalary()); // 50000.0
  System.out.println(ed.getVacationForm()); // pink
  ```

Polymorphism and 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("v.days: " + empl.getVacationDays());
          System.out.println("v.form: " + empl.getVacationForm());
      }
  }
  ```
  
  Output:
  
  ```
  salary: 50000.0
  v.days: 15
  v.form: pink
  ```

Polymorphism and arrays

- Arrays of superclass types can store any subtype as elements.
  
  ```java
  public class EmployeeMain2 {
      public static void main(String[] args) {
          Employee[] e = { new Lawyer(),
                          new Secretary(),
                          new Marketer(),
                          new LegalSecretary() };
          for (int i = 0; i < e.length; i++) {
              System.out.println("salary: " + e[i].getSalary());
              System.out.println("v.days: " + e[i].getVacationDays());
              System.out.println("v.form: " + e[i].getVacationForm());
          }
      }
  }
  ```
  
  Output:
  
  ```
  salary: 50000.0
  v.days: 15
  salary: 50000.0
  v.days: 10
  salary: 60000.0
  v.days: 10
  salary: 55000.0
  v.days: 10
  ```
Polymorphism problems

- 4-5 classes with inheritance relationships are shown.
- A client program calls methods on objects of each class.
- You must read the code and determine the client’s output.
- We always put such a question on our midterms!

A polymorphism problem

- What would be the output of the following client code?

```java
Foo[] pity = {new Baz(), new Bar(), new Mumble(), new Foo()};
for (int i = 0; i < pity.length; i++) {
    System.out.println(pity[i]);
    pity[i].method1();
    pity[i].method2();
    System.out.println();
}
```

Diagramming the classes

- Add classes from top (superclass) to bottom (subclass).
- Include all inherited methods.
Finding output with tables

<table>
<thead>
<tr>
<th>method</th>
<th>Foo</th>
<th>Bar</th>
<th>Baz</th>
<th>Mumble</th>
</tr>
</thead>
<tbody>
<tr>
<td>method1</td>
<td>foo 1</td>
<td>foo 1</td>
<td>baz 1</td>
<td>baz 1</td>
</tr>
<tr>
<td>method2</td>
<td>foo 2</td>
<td>bar 2</td>
<td>foo 2</td>
<td>mumble 2</td>
</tr>
<tr>
<td>toString</td>
<td>foo</td>
<td>foo</td>
<td>baz</td>
<td>baz</td>
</tr>
</tbody>
</table>

Polymorphism answer

```java
Foo[] pity = {new Baz(), new Bar(), new Mumble(), new Foo()};
for (int i = 0; i < pity.length; i++) {
    System.out.println(pity[i]);
    pity[i].method1();
    pity[i].method2();
    System.out.println();
}
```

Output:

baz
baz 1
foo 2
foo
foo 1
bar 2
baz
baz 1
mumble 2
foo
foo 1
foo 2

Casting references

• A variable can only call that type's methods, not a subtype's.

```java
Employee ed = new Lawyer();
int hours = ed.getHours(); // ok; it's in Employee
ed.sue(); // compiler error
```

– The compiler's reasoning is, variable `ed` could store any kind of employee, and not all kinds know how to `sue`.

• To use `Lawyer` methods on `ed`, we can type-cast it.

```java
Lawyer theRealEd = (Lawyer) ed;
theRealEd.sue(); // ok
((Lawyer) ed).sue(); // shorter version
```

More about casting

• The code crashes if you cast an object too far down the tree.

```java
Employee eric = new Secretary();
((Secretary) eric).takeDictation("hi"); // ok
((LegalSecretary) eric).fileLegalBriefs(); // exception
```

• You can cast only up and down the tree, not sideways.

```java
Lawyer linda = new Lawyer();
((Secretary) linda).takeDictation("hi"); // error
```

• Casting doesn't actually change the object's behavior. It just gets the code to compile/run.

```java
((Employee) linda).getVacationForm() // pink (Lawyer's)
Run-Time Type Information

- You can check the legality of a cast before you do it:
  Lawyer linda = new Lawyer();
  if (linda instanceof Secretary) {
    ((Secretary) linda).takeDictation("hi");
  }
- It's generally best to avoid casts as much as possible.
- In many cases, reliance on `instanceof` can be replaced by proper use of polymorphism.

How Does Inheritance Work?

- When class B extends class A, the fields in class A are a subset:
  ```
  Employee class
  ed
  (Employee) ed
  Secretary class
  double Salary
  String VacationForm
  int VacationDays
  double TypingSpeed
   Employee class
  ed
  ```
- Therefore, every Secretary can be treated as an Employee by only looking at the first three fields

How Does Polymorphism Work?

- A subclass has all the methods of the superclass:
  ```
  Method Table / vtable
  Special Class Data
  String toString()
  String VacationForm
  int VacationDays
  double TypingSpeed
  ```
- Overriding a method changes the entry in the method table for this class.
- When we cast a Secretary as an Employee, the method table is unchanged: `toString()` and `getSalary()` still point to the Secretary-specific code

Real Interview Question

```java
/* What does the following program print? */
public class Test {
  public boolean equals( Test other ) {
    System.out.println( "Inside of Test.equals" );
    return false;
  }
  public static void main( String[] args ) {
    Object t1 = new Test();
    Object t2 = new Test();
    Test t3 = new Test();
    Object o1 = new Object();
    System.out.println("1");  t1.equals(t2);
    System.out.println("2");  t1.equals(t3);
    System.out.println("3");  t3.equals(o1);
    System.out.println("4");  t3.equals(t3);
    System.out.println("5");  t3.equals(t2);
  }
}
```
Polymorphism vs. Overloading

• **Overloading**: Two or more methods with *different parameters* and the same name. Which method to call is chosen *statically* (at compile time).

  ```java
  public void add(int value);
  public void add(int value, int index);
  public void add(ArrayIntList list);
  ```

• **Polymorphism**: Related classes define a method with the same name and parameters. Which method to call is chosen *dynamically* (at runtime).