

# Building Java Programs

Chapter 9

Lecture 9-3: Polymorphism

**reading: 9.2**

self-check: #5-9

# 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.
  - `CritterMain` can interact with any type of critter.
    - Each one moves, etc. in its own way.

# Coding with polymorphism

- A variable of type  $T$  can hold an object of any subclass of  $T$ .

```
Employee ed = new Lawyer();
```

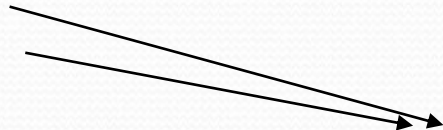
- You can call any methods from `Employee` on `ed`.
  - You can *not* call any methods specific to `Lawyer` (e.g. `sue`).
- When a method is called on `ed`, it behaves as a `Lawyer`.

```
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.

```
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());  
        System.out.println();  
    }  
}
```

A diagram consisting of two arrows. The first arrow originates from the **printInfo(lisa);** line in the `main` method and points to the `printInfo` method signature. The second arrow originates from the `printInfo(steve);` line and also points to the same `printInfo` method signature. This illustrates that both `Lawyer` and `Secretary` objects are passed to the same `printInfo` method, which is designed to accept an `Employee` parameter.

## OUTPUT:

```
salary = 50000.0  
vacation days = 21  
vacation form = pink
```

```
salary = 50000.0  
vacation days = 10  
vacation form = yellow
```

# Polymorphism and arrays

- Arrays of superclass types can store any subtype as elements.

```
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();  
        }  
    }  
}
```

## 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 place such a question on our final exams!

# A polymorphism problem

- Assume that the following four classes have been declared:

```
public class Foo {  
    public void method1() {  
        System.out.println("foo 1");  
    }  
  
    public void method2() {  
        System.out.println("foo 2");  
    }  
  
    public String toString() {  
        return "foo";  
    }  
}  
  
public class Bar extends Foo {  
    public void method2() {  
        System.out.println("bar 2");  
    }  
}
```

# A polymorphism problem

```
public class Baz extends Foo {
    public void method1() {
        System.out.println("baz 1");
    }
    public String toString() {
        return "baz";
    }
}

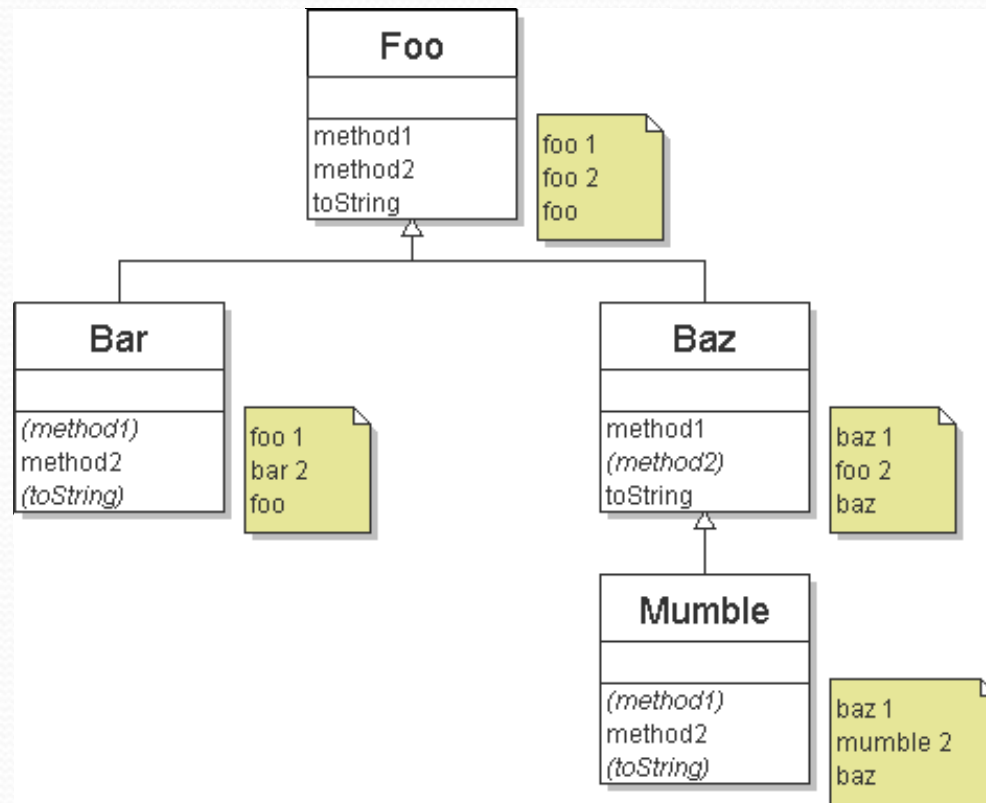
public class Mumble extends Baz {
    public void method2() {
        System.out.println("mumble 2");
    }
}
```

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

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

# Diagramming the classes

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



# Finding output with tables

<b>method</b>	<b>Foo</b>	<b>Bar</b>	<b>Baz</b>	<b>Mumble</b>
method1	foo 1	<i>foo 1</i>	baz 1	<i>baz 1</i>
method2	foo 2	bar 2	<i>foo 2</i>	mumble 2
toString	foo	<i>foo</i>	baz	<i>baz</i>

# Polymorphism answer

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

- **Output:**

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