hi

Building Java Programs

Chapter 9
Lecture 9-2: Polymorphism

reading: 9.3

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.

2

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 the `Employee` class on `ed`.

- 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
  ```

3

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("v.days: " + empl.getVacationDays());
          System.out.println("v.form: " + empl.getVacationForm());
          System.out.println();
      }
  }
  ```

  OUTPUT:
  ```
  salary: 50000.0        salary: 50000.0
  v.days: 15             v.days: 10
  v.form: pink           v.form: yellow
  ```

4

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
  ```

5

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 final exams!

6

bye

hi

A polymorphism problem

- Suppose that the following four classes have been declared:

```java
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");
    }
}
```

7

A polymorphism problem

```java
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?

```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();
}
```

8

Diagramming the classes

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

9

Finding output with tables

method	Foo	Bar	Baz	Mumble
method1	foo 1	foo 1	baz 1	baz 1
method2	foo 2	bar 2	foo 2	mumble 2
toString	foo	foo	baz	baz

10

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
```

11

Another problem

- The order of the classes is jumbled up.
- The methods sometimes call other methods (tricky!).

```java
public class Lamb extends Ham {
    public void b() {
        System.out.print("Lamb b   ");
    }
}
public class Ham {
    public void a() {
        System.out.print("Ham a   ");
        b();
    }
    public void b() {
        System.out.print("Ham b   ");
    }
    public String toString() {
        return "Ham";
    }
}
```

12

bye

2

hi

Another problem 2

```
public class Spam extends Yam {
    public void b() {
        System.out.print("Spam b   ");
    }
}
public class Yam extends Lamb {
    public void a() {
        System.out.print("Yam a    ");
        super.a();
    }

    public String toString() {
        return "Yam";
    }
}
```
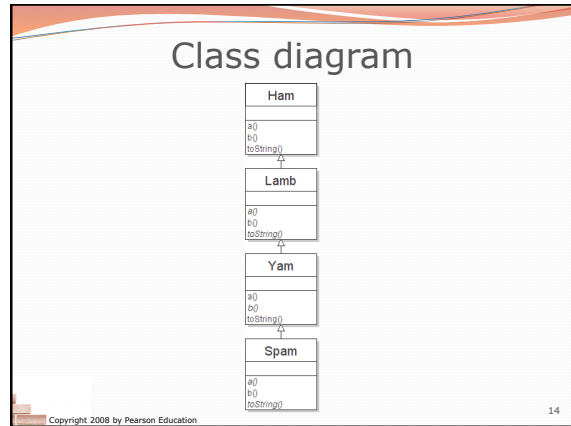
- What would be the output of the following client code?

```
Ham[] food = {new Lamb(), new Ham(), new Spam(), new Yam()};
for (int i = 0; i < food.length; i++) {
    System.out.println(food[i]);
    food[i].a();
    System.out.println();        // to end the line of output
    food[i].b();
    System.out.println();        // to end the line of output
    System.out.println();
}
```

13

Class diagram


```
Ham
────────
a()
b()
toString()
   △
   │
Lamb
────────
a()
b()
toString()
   △
   │
Yam
────────
a()
b()
toString()
   △
   │
Spam
────────
a()
b()
toString()
```

14

Polymorphism at work

- `Lamb` inherits `Ham`'s `a`. `a` calls `b`. But `Lamb` overrides `b`...

```
public class Ham {
    public void a() {
        System.out.print("Ham a    ");
        b();
    }
    public void b() {
        System.out.print("Ham b    ");
    }
    public String toString() {
        return "Ham";
    }
}

public class Lamb extends Ham {
    public void b() {
        System.out.print("Lamb b    ");
    }
}
```

- `Lamb`'s output from `a`:
  ```
  Ham a    Lamb b
  ```

15

The table

method	Ham	Lamb	Yam	Spam
a	Ham a b()	*Ham a* *b()*	Yam a Ham a b()	*Yam a* *Ham a* *b()*
b	Ham b	Lamb b	Lamb b	Spam b
toString	Ham	*Ham*	Yam	*Yam*

16

The answer

```
Ham[] food = {new Lamb(), new Ham(), new Spam(), new Yam()};
for (int i = 0; i < food.length; i++) {
    System.out.println(food[i]);
    food[i].a();
    food[i].b();
    System.out.println();
}
```

- Output:
  ```
  Ham
  Ham a    Lamb b
  Lamb b
  Ham
  Ham a    Ham b
  Ham b
  Yam
  Yam a    Ham a    Spam b
  Spam b
  Yam
  Yam a    Ham a    Lamb b
  Lamb b
  ```

17

Casting references

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

  ```
  Employee ed = new Lawyer();
  int hours = ed.getHours();  // ok; this is 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.

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

  ((Lawyer) ed).sue();                    // shorter version
  ```

18

bye

3

hi

More about casting

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

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

// (Secretary object doesn't know how to file briefs)
```

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

```
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.

```
((Employee) linda).getVacationForm()     // pink (Lawyer's)
```

19

bye