

Polymorphism

zyBook Chap 10.5

Polymorphism

- poly-morphism → **many forms**
- A class can implement an **inherited** method in **its own way**
 - That is, **override** the inherited method. Such as toString() and equals()
- Allows a **variable** of a **superclass type** to **refer to** an object of **one of its subclasses**
 - `<SuperclassName> <objName> = new <SubclassName>();`

Polymorphism – Reference Types

Polymorphism refers to **determining which overridden method to execute depending on data types**

- `<SuperclassName> <objName> = new <SubclassName>();`

```
Person p1 = new Person(); // Declared as a Person, refers to a Person
System.out.println(p1.toString()); // the Person version toString() will be called
```

```
Person p2 = new Student(); // Declared as a Person, refers to a Student
System.out.println(p2.toString()); // the Student version toString() will be called
```

```
You-Know-Who (age: 0)
```

```
You-Know-Who (age: 0) is a Student
```

Why Polymorphism

- The container class (a special component that can hold the gathering of the components – not covered in CS1101) can use the parent type in the composition relationship (has-a relationship)

```
// This Person array allows reference to all these  
// different types of objects that are instances of its subclasses  
Person[] newToVandy = new Person[3];  
  
newToVandy[0] = new Student(18, "new Stu", "nS");  
newToVandy[1] = new Employee(28, "new Emp", "nE");  
newToVandy[2] = new Faculty(38, "new Fac", "nF");  
System.out.println(Arrays.toString(newToVandy));
```

[new Stu (age: 18) is a Student, new Emp (age: 28) is an Employee, new Fac (age: 38) is an Employee – Faculty]

Polymorphism – Reference Types


```
Person p1 = new Person(); // Declared as a Person, refers to a Person
System.out.println(p1.toString()); // the Person version toString() will be called
```

```
Person p2 = new Student(); // Declared as a Person, refers to a Student
System.out.println(p2.toString()); // the Student version toString() will be called
```

```
System.out.println(p2.getCourse());
```



```
System.out.println(p2.getCourse());
```

 The method getCourse() is undefined for the type Person

Though a Person object can **refer** to a Student object, and **perform** the **overridden version** of certain methods in Student Class (e.g., toString()), a Person object **CANNOT perform the methods ONLY** in the Student Class

```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```



```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

Does Tulip have a toString()?
No

Does Tulip explicitly have a
toString()?
No
Look into its superclass

Q: What's the output
of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

pretty[0]: Rose Lily

Q: What's the output of the client code? →

```

Violet[] pretty = { new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

pretty[0]: Rose Lily

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```



```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

pretty[0]: Rose Lily

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```



```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

```
pretty[0]: Rose Lily
Tulip 1
```

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

```
pretty[0]: Rose Lily
Tulip 1
```

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

```

public class Violet {
    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Lily extends Violet {
    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Rose extends Lily {
    public String toString() {
        return "Rose " + super.toString();
    }
}

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

```

```
pretty[0]: Rose Lily
Tulip 1
```

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

```

public class Violet {
    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Lily extends Violet {
    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Rose extends Lily {
    public String toString() {
        return "Rose " + super.toString();
    }
}

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

```

```

pretty[0]: Rose Lily
Tulip 1
Lily 2

```

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

Which method1 will be printed?
Lily's version or Tulip's version?

```

pretty[0]: Rose Lily
Tulip 1
Lily 2

```

Q: What's the output
of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

Since the pretty[0] refers to a Tulip, and Tulip Class contains an overridden method1()

```

pretty[0]: Rose Lily
Tulip 1
Lily 2

```

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

Since the pretty[0] refers to a Tulip, and Tulip Class contains an overridden method1()

```

pretty[0]: Rose Lily
Tulip 1
Lily 2
Tulip 1

```

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

```

pretty[0]: Rose Lily
Tulip 1
Lily 2
Tulip 1
=====

```

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```



```

public class Violet {

    public void method1() {
        System.out.println("Violet 1 ");
    }

    public void method2() {
        System.out.println("Violet 2 ");
    }

    public String toString() {
        return "Violet";
    }
}

public class Rose extends Lily {

    public String toString() {
        return "Rose " + super.toString();
    }
}

```

```

public class Lily extends Violet {

    public void method1() {
        super.method1();
        System.out.println("Lily 1 ");
    }

    public void method2() {
        System.out.println("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}

public class Tulip extends Rose {

    public void method1() {
        System.out.println("Tulip 1 ");
    }
}

```

```

pretty[0]: Rose Lily
Tulip 1
Lily 2
Tulip 1
=====
pretty[1]: Lily
Violet 1
Lily 1
Lily 2
Violet 1
Lily 1
=====
pretty[2]: Violet
Violet 1
Violet 2
=====
pretty[3]: Rose Lily
Violet 1
Lily 1
Lily 2
Violet 1
Lily 1
=====

```

Q: What's the output of the client code? →

```

Violet[] pretty = {new Tulip(), new Lily(), new Violet(), new Rose() };

for (int i = 0; i < pretty.length; ++i) {
    System.out.println("pretty[" + i + "]: " + pretty[i]);
    pretty[i].method1();
    pretty[i].method2();
    System.out.println("=====");
}

```

Summary – Four Main Principles of OOP

1. **Abstraction**

To simplify reality and focus only on the properties and external behaviors rather than inner details

2. **Encapsulation**

Hiding the implementation details (data and the programs that manipulate the data) of an object from the clients of the object

3. **Inheritance**

A class can derive its methods and properties from another class

4. **Polymorphism**

Overriding methods among subclasses, and determining which overridden method to execute depending on data types