Inheritance

Part 2

Learning Objectives

1. Learn how to use the **super** keyword
2. Learn method **overrides**.

Agenda

- Recall Inheritance concepts
- Review/learn super keyword
- Learn method overrides
- Coding time: practice

Quick Recall

- Inheritance
- Superclass
- Subclass
- What can be inherited by a subclass?
- Java syntax: subclass, superclass

IS-A and HAS-A

IS-A Relationship:

Apple IS-A Fruit suggests that Apple is a subclass of Fruit. Apple can do everything that Fruit can do. Apple can be used as a Fruit

HAS-A Relationship:

Room HAS-A Table suggests that Room contains at least one Table inside of it. Table could be an instance variable inside of Room.

Subclass Constructors

Subclasses do not inherit the constructor from the superclass.

This means that a subclass needs to have its own constructor, or use the default constructor.

Subclass constructor calls superclass

Subclass constructors must call the superclass constructor.

When a subclass is created, a superclass is also created. The subclass must make a call to the superclass constructor.

```
public class Student extends Person{
    private int grade;
    public Student (String fName, String lName, String addr, int gr) {
            super(fName, lName, addr); // Call to superclass constructor
            grade = gr;
    }
}
```

Subclass constructor calls superclass

To call the superclass constructor, we use the keyword **super on the first line of our subclass constructor,** then pass it the required parameters.

The actual parameters passed in the call to the superclass provide values that the superclass constructor can use to initialize the object's instance variables.

```
public class Student extends Person{
        private int grade;
        public Student (String fName, String lName, String addr, int gr) {
                super(fName, lName, addr); // Call to superclass constructor
                grade = gr;
        }
}
```

Special note on subclass constructor

When a subclass **constructor does not explicitly call a superclass constructor** using super, Java inserts **a call to the superclass no-argument constructor** (default constructor).

If a **no-argument constructor doesn't exist**, then the program will **not compile** without an explicit call to a constructor.

Calling superclass methods from a subclass

When we create an object and want to call a method for that object we do so using the object name and method. For example:

```
Square box =  new Square(5);

box.area();
```

How do we do this if we don't have a name for our object in the subclass?

```
public class Square extends Rectangle{
        ……
        public String toString(){
                return "Square area: " + …….; // ???
        }
}
```

Calling superclass methods from a subclass

```
public class Square extends Rectangle{
        ……
        public String toString(){
                return "Square area: " + super.area();
                // or
                // return "Square area: " + area();
        }
}
```

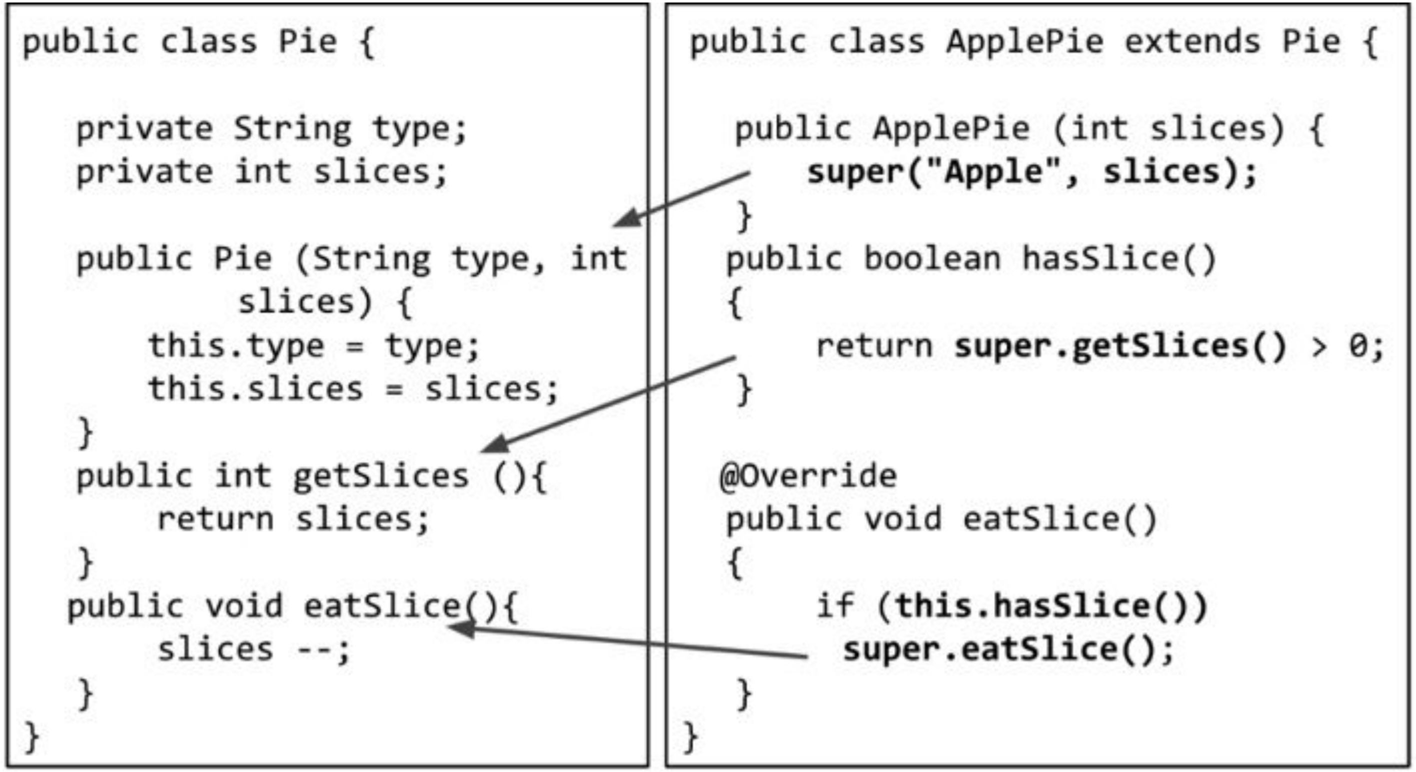
Similar to **this** keyword

The keyword **super** is similar to the keyword **this**.

- **this** refers to the current object and we can call methods using: **this**.myMethod(arguments)

- **super** refers to the superclass object and we can call methods using **super**.myMethod(arguments)

Example using **super**



```
public class Pie {

   private String type;
   private int slices;

   public Pie (String type, int
            slices) {
       this.type = type;
       this.slices = slices;
   }
   public int getSlices (){
        return slices;
   }
  public void eatSlice(){
        slices --;
   }
}
```

```
public class ApplePie extends Pie {

   public ApplePie (int slices) {
        super("Apple", slices);
   }
  public boolean hasSlice()
  {
        return super.getSlices() > 0;
  }

  @Override
  public void eatSlice()
  {
        if (this.hasSlice())
          super.eatSlice();
  }
}
```

super Keyword Recap

The keyword super can be used to call a superclass's constructors and methods.

The superclass methods can be called in a subclass by using the keyword super with the method name and passing appropriate parameters.

What if we don't want to use a superclass method?

There are times when we may not want to inherit a method from the parent.

Person Class

```
public  String toString(){
        return name + " was born " + birthday;
}
```

Student Class

```
public  String toString(){
        return super.getName() + " is in grade " + grade;
}
```

Method Overrides

Method Overrides allow a subclass to redefine a method instead of inheriting that method from the superclass.

Method Overrides occurs when a public method in a subclass has the **same method signature** as a method in the superclass.

When a method override occurs, the subclass method will be called.

Person Class:

```
public  String toString(){
        return name + " was born" + birthday;
}
```

Student Class:

```
public  String toString(){
        return super.getName() + " is in grade " + grade;
}
```

Override Notation

When we override a method, we can use the **@Override** notation.

Student Class:

```
@Override
public  String toString(){
        return super.getName() + " is in grade " + grade;
}
```

Override Notation

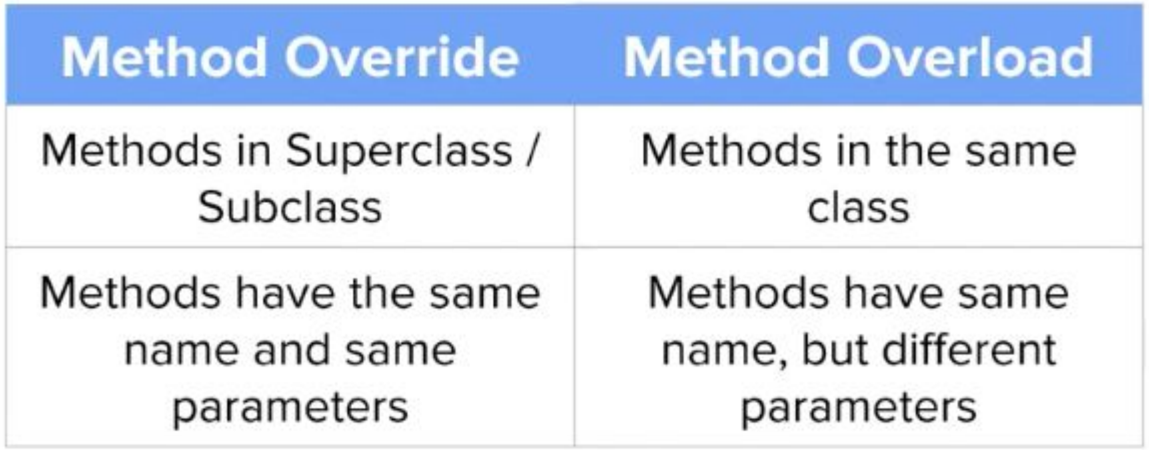
@Override is not required, a method with the same signature as the superclass will still be overridden.

Using @Override notion is best practice for two main reasons:

1. It helps with debugging. When you use @Override the compiler double checks that you correctly override the method from the superclass (without @Override the compiler only checks syntax).
2. It makes your code more readable since it becomes clear that you are overriding a superclass method.

Override vs. Overload

Similar terms, but different implementations.



Method Override	Method Overload
Methods in Superclass / Subclass	Methods in the same class
Methods have the same name and same parameters	Methods have same name, but different parameters

Class Hierarchy Considerations

When calling methods for an object, Java first looks in that object class. If it is defined in that class, it will use that method, otherwise it will look to the superclass for the method.

Any method that is called must be defined within its own class or its superclass. Otherwise, Java will throw an error.

Overriding Methods Recap

- A method called by an object needs to be defined someplace in the class hierarchy.

- A subclass inherits all public methods from the superclass. It can then add additional fields and/or methods, or modify them with an override.

- Method overriding occurs when a public instance method in a subclass has the same method signature as a public instance method in the superclass.

- You cannot override a private or static method in Java. If you create a similar method with same return type and same method arguments in child class then it will hide the super class method; this is known as **method hiding**.