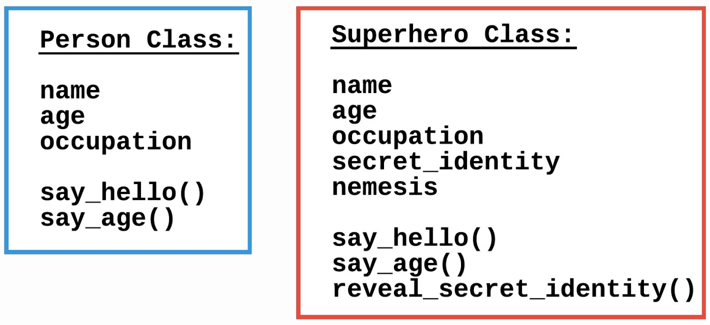
Learning Objectives - Extending & Overriding

- **Define the terms extending and overriding**
- **Extend the superclass with new method**
- **Override methods from the superclass with new functionality**
- **Annotate a method when overriding it**

Extending a Class

Extending the Constructor

The idea of inheritance is to borrow from a superclass and then add on functionality. Up until now, we have talked about borrowing from a superclass. The process of adding functionality to a subclass is known as either extending or overriding. Extending a class means that new attributes and methods are given to the subclass.



Superhero and Person Classes

The code below will first call upon the superclass constructor (using `super`) to create the attributes `name`, `age`, and `occupation`. The constructor is extended when the attribute `secretIdentity` is added to the `Superhero` class.

```java
//add class definitions below this line

class Superhero extends Person {
  private String secretIdentity;

  public Superhero(String na, int a, String o, String s) {
    super(na, a, o);
    secretIdentity = s;
  }

  public String getSecretIdentity() {
    return secretIdentity;
  }

  public void setSecretIdentity(String newIdentity) {
    secretIdentity = newIdentity;
  }
}

//add class definitions above this line
```

Instantiate a `Superhero` object and print out each of the attributes. You should see the three attributes from the `Person` class as well as the new attribute `secretIdentity`.

```java
    //add code below this line

    Superhero hero = new Superhero("Spider-Man", 17, "student",
        "Peter Parker");
    System.out.println(hero.getName());
    System.out.println(hero.getAge());
    System.out.println(hero.getOccupation());
    System.out.println(hero.getSecretIdentity());

    //add code above this line
```

▼ **Inheritance is a One-Way Street**

Inheritance shares attributes and methods from the superclass to the subclass. When a subclass is extended, it cannot share the new additions with their superclass. In the code above, `Superhero` has access to `name`, but `Person` does not have access to `secretIdentity`.

challenge

Try this variation:

- Rewrite the `Superhero` class so that it extends the `Person` class by adding the attribute `nemesis`, Doc Octopus.
  - ▼ **Solution**

```
//add class definitions below this line

class Superhero extends Person {
  private String secretIdentity;
  private String nemesis;

  public Superhero(String na, int a, String o, String s,
        String ne) {
    super(na, a, o);
    secretIdentity = s;
    nemesis = ne;
  }

  public String getSecretIdentity() {
    return secretIdentity;
  }

  public void setSecretIdentity(String newIdentity) {
    secretIdentity = newIdentity;
  }

  public String getNemesis() {
    return nemesis;
  }

  public void setNemesis(String newNemesis) {
    nemesis = newNemesis;
  }
}

//add class definitions above this line
```

Now instantiate a `Superhero` object and make sure that the new attribute works as expected.

```
    //add code below this line

    Superhero hero = new Superhero("Spider-Man", 17,
        "student", "Peter Parker", "Doc Octopus");
    System.out.println(hero.getName());
    System.out.println(hero.getAge());
    System.out.println(hero.getOccupation());
    System.out.println(hero.getSecretIdentity());
    System.out.println(hero.getNemesis());

    //add code above this line
```

Extending a Class by Adding New Methods

Another way to extend a class is to create new methods (besides getters and setters) that are unique to the subclass. For example, the `sayHello` method will give the superhero's name, but it will not divulge their secret identity. Create the method `revealSecretIdentity` to print the attribute `secretIdentity`.

```java
//add class definitions below this line

class Superhero extends Person {
  private String secretIdentity;
  private String nemesis;

  public Superhero(String na, int a, String o, String s, String
        ne) {
    super(na, a, o);
    secretIdentity = s;
    nemesis = ne;
  }

  public String getSecretIdentity() {
    return secretIdentity;
  }

  public void setSecretIdentity(String newIdentity) {
    secretIdentity = newIdentity;
  }

  public String getNemesis() {
    return nemesis;
  }

  public void setNemesis(String newNemesis) {
    nemesis = newNemesis;
  }

  public void revealSecretIdentity() {
    System.out.println("My real name is " + secretIdentity +
        ".");
  }
}

//add class definitions above this line
```

Now test out the newly added method.

```java
    //add code below this line

    Superhero hero = new Superhero("Spider-Man", 17, "student",
        "Peter Parker", "Doc Octopus");
    hero.revealSecretIdentity();

    //add code above this line
```

challenge

Try this variation:

- Create the method `sayNemesis` that prints the string:
  `My nemesis is Doc Octopus..`

  ▼ **Solution**

```java
//add class definitions below this line

class Superhero extends Person {
  private String secretIdentity;
  private String nemesis;

  public Superhero(String na, int a, String o, String s,
        String ne) {
    super(na, a, o);
    secretIdentity = s;
    nemesis = ne;
  }

  public String getSecretIdentity() {
    return secretIdentity;
  }

  public void setSecretIdentity(String newIdentity) {
    secretIdentity = newIdentity;
  }

  public String getNemesis() {
    return nemesis;
  }

  public void setNemesis(String newNemesis) {
    nemesis = newNemesis;
  }

  public void revealSecretIdentity() {
    System.out.println("My real name is " + secretIdentity +
        ".");
  }

  public void sayNemesis() {
    System.out.println("My nemesis is " + nemesis + ".");
  }
}

//add class definitions above this line
```

Now invoke the `sayNemesis` method.

```java
//add code below this line

Superhero hero = new Superhero("Spider-Man", 17,
    "student", "Peter Parker", "Doc Octopus");
hero.sayNemesis();

//add code above this line
```

Method Overriding

Overriding a Method

Extending a class means adding new attributes or methods to the subclass. Another way to add new functionality to a subclass is through method overriding. Overriding a method means to inherit a method from the superclass, keep its name, but change the contents of the method.

Extend the `Superhero` class by overriding the `sayHello`. Add this method to the `Superhero` class. Remember, the `name` attribute is part of the superclass, so you need to use the `getName` method to access this attribute.

```java
public void sayHello() {
  System.out.println("My name is " + getName() + ", and
      criminals fear me.");
}
```

Instantiate a `Superhero` object and call the `sayHello` method.

```java
  //add code below this line

  Superhero hero = new Superhero("Storm", 30, "Queen of
      Wakanda", "Ororo Munroe", "Shadow King");
  hero.sayHello();

  //add code above this line
```

▼ **Differentiating Overriding and Extending**

The difference between extending and overriding can be slight. Both approaches are used to make a subclass unique from the superclass. Overriding deals with changing a pre-existing method from the superclass, while extending deals with adding new methods and attributes.

challenge

Try this variation:

- Override the `sayAge` method so that it prints the string: `Young or old, I will triumph over evil.`

  ▼ **Solution**

  Add the following method to the `Superhero` class:

```
public void sayAge() {
  System.out.println("Young or old, I will triumph over
      evil.");
}
```

Call the method to verify it works as expected.

```
  //add code below this line

  Superhero hero = new Superhero("Storm", 30, "Queen of
      Wakanda", "Ororo Munroe", "Shadow King");
  hero.sayHello();
  hero.sayAge();

  //add code above this line
```

What Happens When You Override a Method?

If you can override a method from the superclass, what happens to the original method? Java defaults to the instance. So `hero.sayHello()` will always use the method from the `Superhero` class. But that does not mean you cannot call `sayHello` from the `Person` class. Just as we used the `super` keyword to access the constructor from the superclass, we can use `super` to invoke `sayHello` from the `Person` class. Add the method `oldHello` to the `Superhero` class. Use `super` followed by the method name to tell Java to look in the `Person` class.

```java
public void sayHello() {
  System.out.println("My name is " + getName() + ", and
      criminals fear me.");
}

public void sayAge() {
  System.out.println("Young or old, I will triumph over
      evil.");
}

public void oldHello() {
  super.sayHello();
}
```

```java
  //add code below this line

  Superhero hero = new Superhero("Storm", 30, "Queen of
      Wakanda", "Ororo Munroe", "Shadow King");
  hero.sayHello();
  hero.oldHello();

  //add code above this line
```

challenge

Try this variation:

- Add the method `oldAge` to the `Superhero` class and then call it.

▼ **Solution**

Add the following method to the `Superhero` class.

```
public void oldAge() {
  super.sayAge();
}
```

Call the method to verify the output works as expected.

```
  //add code below this line

  Superhero hero = new Superhero("Storm", 30, "Queen of
      Wakanda", "Ororo Munroe", "Shadow King");
  hero.sayAge();
  hero.oldAge();

  //add code above this line
```

Overriding Annotation

Overriding Annotation

Java allows you to provide an optional annotation when overriding a method. The `@Override` annotation lets the Java compiler know that the following method overrides a method from the superclass. Adding annotations when overriding methods is also helpful when other developers look at your code. Java can flag an issue when method overriding does not work as expected. For example, the class below is supposed to override the `greeting` method, but there is a typo.

```java
//add class definitions below this line

class ClassB extends ClassA {
  public void greting() {
    System.out.println("Hello from Class B");
  }
}

//add class definitions above this line
```

Now create instances of `ClassA` and `ClassB` and call the `greeting` method. You would expect the output to be a greet from each class. However, due to a typo, `ClassB` does not override the `greeting` method. Instead it defines the method `greting`. Java runs the program without a problem, but the output is not correct.

```java
    //add code below this line

    ClassA a = new ClassA();
    ClassB b = new ClassB();

    a.greeting();
    b.greeting();

    //add code above this line
```

If we add the `@Override` annotation to `ClassB`, you will see how Java notices the problem and brings it to your attention. Java expects `ClassB` to override the `greeting` method. Because it does not, the compiler generates an error message. Annotating method overriding helps programmers catch small problems that are otherwise hard to track down.

```
//add class definitions below this line

class ClassB extends ClassA {
  @Override
  public void greting() {
    System.out.println("Hello from Class B");
  }
}

//add class definitions above this line
```

challenge

Try this variation:

- Fix the typo for the `greeting` method in `ClassB`.

▼ **Solution**

```
class ClassB extends ClassA {
  @Override
  public void greeting() {
    System.out.println("Hello from Class B");
  }
}
```

Prohibit Overriding

Prohibit Overriding

In Java, you can override any public method from the superclass. The code below overrides the `greeting` method. But what if you do not want another user to override a method from your class? The `final` keyword will keep users from overriding a method. Notice how the `greeting` method in `ClassA` uses `final`. Copy and paste this code into the IDE and run it.

```
//add class definitions below this line

class ClassA {
  public final void greeting() {
    System.out.println("Hello from Class A");
  }
}

class ClassB extends ClassA {
  public void greeting() {
    System.out.println("Hello from Class B");
  }
}

//add class definitions above this line
```

Instantiate an object of type `ClassB` and call the `greeting` method. This code will not work. Java says that you cannot override a method that uses `final`.

```
    //add code below this line

    ClassB b = new ClassB();
    b.greeting();

    //add code above this line
```

Normally, you would override `greeting` and then use the `super` keyword to access this method in the superclass. Using `final` makes this impossible. Instead of trying to override `greeting`, create the method `greeting2` in `ClassB`.

```
//add class definitions below this line

class ClassA {
  public final void greeting() {
    System.out.println("Hello from Class A");
  }
}

class ClassB extends ClassA {
  public void greeting2() {
    System.out.println("Hello from Class B");
  }
}

//add class definitions above this line
```

Call both `greeting` and `greeting2` with the `b` object. The program should run and print two different greetings.

```
    //add code below this line

    ClassB b = new ClassB();
    b.greeting();
    b.greeting2();

    //add code above this line
```

Extending & Overriding Formative Assessment 1

Extending & Overriding Formative Assessment 2