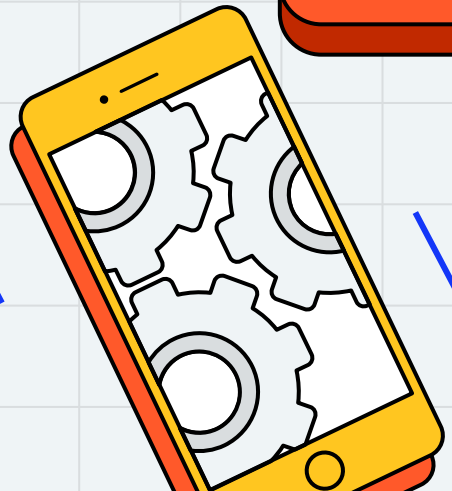
Application Programming



Hend Alkittawi

OOP Concepts

Introduction to Inheritance in Java

INHERITANCE

- When creating a class rather than declaring completely new members you can designate that the new class should **inherit** the members of an existing class.
- The existing class is called the **superclass** and the new class is the **subclass**.
- With inheritance the instance variables and methods that are the same for all the classes in the hierarchy are declared in a superclass.
- In inheritance, a new class is created by acquiring an existing class's members and possibly embellishing them with **new** or **modified** capabilities.

```java
public class Animal {

   private String name;

   public String getName() {
     return name;
   }

   public void setName(String name) {
     this.name = name;
   }

   public void eat() {
     System.out.println(getName() +
                              " eats food.");
   }
}
```

```java
public class Dog extends Animal {

   private String breed;

   public String getBreed() {
     return breed;
   }

   public void setBreed(String breed) {
     this.breed = breed;
   }

   public void bark() {
     System.out.println("The dog barks.");
   }
}
```

```java
public class InheritanceDemo {

  public static void main(String[] args){

    Animal animal = new Animal();
    animal.eat();

    Dog dog = new Dog();
    // setName() is an inherited method
    dog.setName("Buddy");

    // setBreed() is a method of Dog class
    dog.setBreed("Golden Retriever");

    // eat() is an inherited method
    dog.eat();

    // bark() is a method of Dog class
    dog.bark();
  }
}
```

INHERITANCE

- The **direct** superclass is the superclass from which the subclass explicitly inherits.
- An **indirect** superclass is any class above the direct superclass in the class hierarchy, which defines the inheritance relationships among classes.
- In Java, the class hierarchy begins with a class **Object** which every class in Java directly or indirectly extends.
- Java supports only **single inheritance** in which each class is derived from exactly one direct superclass.

INHERITANCE

- Inheritance and constructors
  - Constructors are not inherited, a superclass’s constructors are still available to be called by subclasses.
  - Java requires that the first task of any subclass constructor is to call its direct superclass’s constructor to ensure that the instance variables inherited from the superclass are initialized properly.
  - Superclass constructor call syntax: keyword **super** followed by a set of parentheses containing the super class constructor arguments which are you used to initialize the super class instance variables.

```java
public class Animal {

   private String name;

   public Animal(String name) {
     this.name = name;
   }

   public String getName() {
     return name;
   }

   public void setName(String name) {
     this.name = name;
   }

   public void eat() {
     System.out.println(getName() +
                            " eats food.");
   }
}
```

```java
public class Dog extends Animal {

   private String breed;

   public Dog(String name, String breed) {
     super(name);
     this.breed = breed;
   }

   public String getBreed() {
     return breed;
   }

   public void setBreed(String breed) {
     this.breed = breed;
   }

   public void bark() {
     System.out.println("The dog barks.");
   }
}
```

```java
public class InheritanceDemo {

  public static void main(String[] args){

    Animal animal = new Animal("Hazel");
    animal.eat();

    Dog dog = new Dog("A Name", "A Breed");

    // eat() is an inherited method
    dog.eat();

    // bark() is a method of Dog class
    dog.bark();
  }
}
```

INHERITANCE

- A subclass can **add** its own fields and methods; it is more specific than its superclass.
- A subclass exhibits the behavior of its superclass and can **modify** these behaviors so that they operate appropriately for the subclass. A subclass can customize methods that it inherits from its superclass to do this the subclass **overrides**/redefines the superclass method with an appropriate implementation.
- To override a superclass method in a subclass, the subclass must declare a method with the **same signature as the superclass method.**
- When a subclass method overrides an inherited superclass method, the superclass version of the method can be accessed from the subclass by preceding the super-class method name with the keyword **super** and a DOT (.) separator.

```java
public class Animal {

   private String name;

   public Animal(String name) {
     this.name = name;
   }

   public String getName() {
     return name;
   }

   public void setName(String name) {
     this.name = name;
   }

   public void eat() {
     System.out.println(getName() +
                            " eats food.");
   }

   @Override
   public String toString() {
     return "Name: " + getName();
   }
}
```

```java
public class Dog extends Animal {

   private String breed;

   public Dog(String name, String breed) {
     super(name);
     this.breed = breed;
   }

   public String getBreed() {
     return breed;
   }

   public void setBreed(String breed) {
     this.breed = breed;
   }

   public void bark() {
     System.out.println("The dog barks.");
   }

   @Override
   public String toString() {
     return super.toString() + " Breed: "
                                + getBreed();
   }
}
```

```java
public class InheritanceDemo {

  public static void main(String[] args){

    Animal animal = new Animal("Hazel");
    animal.eat();

    Dog dog = new Dog("A Name", "A Breed");
    // eat() is an inherited method
    dog.eat();

    // bark() is a method of Dog class
    dog.bark();

    // toString() in Dog redefines the
    // behavior of toString() in Animal
    String s = dog.toString();
  }
}
```

PUBLIC, PRIVATE AND PROTECTED KEYWORDS

- A class's **public** members are accessible **wherever the program has reference to an object** of that class or one of its subclasses.
- A class's **private** members are accessible **only within the class itself**.
- Using **protected** access modifier offers an intermediate level of access between public and private; a superclass's protected members **can be accessed by members of that superclass, by members of its subclasses and by members of other classes in the same package**.

PUBLIC, PRIVATE AND PROTECTED KEYWORDS

- Public members of the superclass become public members of the subclass and protected members of the superclass become protected members of the subclass.

- Methods of a subclass cannot directly access private members of their superclass. Declaring private instance variables helps you test, debug and correctly modify systems.

```java
package inheritance;
public class Animal {

   private String name;
   public String aString;
   public String publicString;
   protected String protectedString;

   public Animal(String name) {
     this.name = name;
   }

   public String getName() {
     return name;
   }

   public void setName(String name) {
     this.name = name;
   }

   public void eat() {
     System.out.println(getName() +
                            " eats food.");
   }

   @Override
   public String toString() {
     return "Name: " + getName();
   }

   private void animalMethod() {
    // method body
   }
}
```

```java
package inheritance;
public class Dog extends Animal {

   private String breed;

   public Dog(String name, String breed) {
     super(name);
     this.breed = breed;
   }

   public String getBreed() {
     return breed;
   }

   public void setBreed(String breed) {
     this.breed = breed;
   }

   public void bark() {
     System.out.println("The dog barks.");
   }

   private void testAccess() {
     // String s0 = name; // invalid
     String s1 = getName(); // indirect access
     String s2 = publicString; // insecure
     String s3 = protectedString; // valid
   }

   @Override
   public String toString() {
     return super.toString() + " Breed: "
                                + getBreed();
   }
}
```

```java
package inheritance;
public class InheritanceDemo {

  public static void main(String[] args){

    Animal animal = new Animal("Hazel");
    animal.eat();
 // animal.animalMethod(); // inavlid
 // animal.bark(); // invalid
 // animal.name = "some string"; // invalid
    animal.publicString = "some string";
    animal.protectedString = "some string";

    Dog dog = new Dog("A Name", "A Breed");
    // eat() is an inherited method
    dog.eat();

    // bark() is a method of Dog class
    dog.bark();

    // toString() in Dog redefines the
    // behavior of toString() in Animal
    String s = dog.toString();

 // dog.name = "some string"; // invalid
    dog.publicString = "some string";
    dog.protectedString = "some string";
  }
}
```

CODE DEMO

- Create classes to demo inheritance concepts!

THANK YOU!

DO YOU HAVE ANY QUESTIONS?

hend.alkittawi@utsa.edu

By Appointment

Online