Implementing Polymorphism

Estimated time needed: 30 minutes

In this lab, you will learn how to implement polymorphism in a Java class.

You are currently viewing this lab in a Cloud based Integrated Development Environment (Cloud IDE). It is a fully-online integrated development environment that is pre-installed with JDK 21, allowing you to code, develop, and learn in one location.

Learning Objectives

After completing this lab, you will be able to:

- Create a class as a superclass
- Create subclasses that inherit from the superclass
- Understand how to use polymorphism to create instances of subclass
- Override the methods inherited from the superclass in the subclass.
- Obtain input from user and create objects
- Store the objects of subclasses in an array initialized as that of superclass

Create Superclass Animal

You will create a class `Animal` with methods that are common to all animals. The `Animal` class will be the superclass when it is inherited by other classes.

1. Create a project directory by running the following command.

   ```
   mkdir my_poly_proj
   ```

2. Run the following code to create the directory structure.

   ```
   mkdir -p my_poly_proj/src
   mkdir -p my_poly_proj/classes
   mkdir -p my_poly_proj/test
   cd my_poly_proj
   ```

3. Now, create a file named `Animal.java` inside the src directory.

```
touch /home/project/my_poly_proj/src/Animal.java
```

4. Click the following button to open the file for editing.

Open **Animal.java** in IDE

5. Read each statement in the following program and understand all the methods that are implemented in `Animal` class.

Paste the following content in `Animal.java`.

```
public class Animal {
    private String name;
    public Animal(String name) {
        this.name = name;
    }
    public String sound() {
        return null;
    }
    public String toString() {
        return name.concat(" says ").concat(sound());
    }
}
```

6. Compile the Java program, specifying the destination directory as the `classes` directory that you created.

```
javac -d classes src/Animal.java
```

7. Set the `CLASSPATH` variable.

```
export CLASSPATH=$CLASSPATH:/home/project/my_poly_proj/classes
```

8. Paste the following content in `Animal.java` after the existing class definition.

```
class Dog extends Animal {
    public Dog(String name) {
        super(name);
    }
    public String sound() {
        return "Woof";
    }
}
class Cat extends Animal {
    public Cat(String name){
        super(name);
    }
    public String sound() {
        return "Meow";
    }
}
class Cow extends Animal {
    public Cow(String name){
        super(name);
    }
    public String sound() {
        return "Moo";
    }
}
```

Notice that you can have only one public class in `.java` file and the name of the file should be the same as the public class. The other classes in the file don't have an access modifier, which means they are using `default` modifier.

9. Compile the classes, specifying the destination directory as the `classes` directory that you created.

```
javac -d classes src/Animal.java
```

Create a program to create Animal objects

1. Now, create a file named `AnimalFarm.java` inside the src directory.

```
touch /home/project/my_poly_proj/src/AnimalFarm.java
```

2. Click the following button to open the file for editing.

Open **AnimalFarm.java** in IDE

3. Read each statement in the following program and understand how the objects of the subclasses are created.

Paste the following content in `AnimalFarm.java`.

```
public class AnimalFarm {
    public static void main(String s[]) {
        Animal animal1 = new Dog("Sami");
        Animal animal2 = new Cat("Hershery");
        Animal animal3 = new Cow("Molly");
        System.out.println("animal1 sound " + animal1.sound());
        System.out.println("animal2 sound " + animal2.sound());
        System.out.println("animal3 sound " + animal3.sound());
    }
}
```

Three objects are created: animal1, animal2, and animal3, each of type Animal. However, they are instantiated with different classes: Dog, Cat, and Cow, respectively.

The key aspect of this code is polymorphism. Although the objects are declared as type Animal, they are actually instances of specific animal classes (Dog, Cat, Cow). When the sound() method is called on each object, the correct implementation is invoked based on the actual object type.

Method Invocation

The sound() method is called on each object, and the corresponding implementation is executed:

- animal1.sound() calls the sound() method implemented in the Dog class.
- animal2.sound() calls the sound() method implemented in the Cat class.
- animal3.sound() calls the sound() method implemented in the Cow class.

3. Compile the Java program, specifying the destination directory as the `classes` directory that you created.

```
javac -d classes src/AnimalFarm.java
```

4. Run the program and observe the output.

```
java AnimalFarm
```

You will see the output as below:

```
animal1 sound Woof
animal2 sound Meow
animal3 sound Moo
```

Create Animal Objects Array

1. Click the following button to open the file `AnimalFarm.java` for editing.

Open **AnimalFarm.java** in IDE

2. Read each statement in the following program and understand how to create an array of Animals of the user's choice. Paste the following content in `AnimalFarm.java`.

```
import java.util.Scanner;
public class AnimalFarm {
    public static void main(String s[]) {
        Scanner scanner = new Scanner(System.in);
        Animal[] animals = new Animal[10];
        int anmlIdx = 0;
        while(true) {
            System.out.println( "Press 1 to view the animals, " +
                                "\n2 to add animals, "+
                                "\nany other key to exit");
            String userAction = scanner.nextLine();
            if (userAction.equals("1")) {
                for(int i=0;i<animals.length;i++) {
                    if(animals[i] != null) {
                        System.out.println(animals[i]);
                    }
                }
            } else if (userAction.equals("2")) {
                if(anmlIdx == 10) {
                    System.out.println("10 animals added already. Cannot add any more animals!");
                    continue;
                }
                System.out.println("Which animal do you want to create? \nPress 1 for dog,"+
                                    "\n2 for cat " +
                                    "\n3 for cow" );
                String animalChoice = scanner.nextLine();
                if (animalChoice.equals("1")) {
                    System.out.println("Enter the name of the dog");
                    String dogName = scanner.nextLine();
                    Animal anmlTmp = new Dog(dogName);
                    animals[anmlIdx++] = anmlTmp;
                } else if (animalChoice.equals("2")) {
                    System.out.println("Enter the name of the cat");
                    String catName = scanner.nextLine();
                    Animal anmlTmp = new Cat(catName);
                    animals[anmlIdx++] = anmlTmp;
                } else if (animalChoice.equals("3")) {
                    System.out.println("Enter the name of the cow");
                    String cowName = scanner.nextLine();
                    Animal anmlTmp = new Cow(cowName);
                    animals[anmlIdx++] = anmlTmp;
                }
            } else {
                break;
            }
        }
    }
}
```

The program enters an infinite loop, which continues until the user chooses to exit.

Inside the loop, the program prompts the user to:

- View the animals (press 1)
- Add animals (press 2)
- Exit (press any other key)

If the user presses 1, the program iterates through the animals array and prints the details of each animal using the toString method in the superclass Animal.

If the user presses 2, the program:

- Checks if the animals array is full. If so, it displays an error message and skips to the next iteration.

- Asks the user to choose the type of animal to add (dog, cat, or cow).
- Based on the user's choice, it prompts for the animal's name and creates a new instance of the corresponding animal class (Dog, Cat, or Cow).
- Adds the new animal to the animals array and increments the anmlIdx variable.

If the user presses any key other than 1 or 2, the program exits the loop and terminates.

3. Compile the Java program, specifying the destination directory as the `classes` directory that you created.

   ```
   javac -d classes src/AnimalFarm.java
   ```

4. Now, run the java program.

   ```
   java AnimalFarm
   ```

Your sample output, which depends on the user input, would appear as below:

```
Press 1 to view the animals,
2 to add animals,
any other key to exit
2
Which animal do you want to create?
Press 1 for dog,
2 for cat
3 for cow
1
Enter the name of the dog
Hershey
Press 1 to view the animals,
2 to add animals,
any other key to exit
2
Which animal do you want to create?
Press 1 for dog,
2 for cat
3 for cow
2
Enter the name of the cat
Sami
Press 1 to view the animals,
2 to add animals,
any other key to exit
2
Which animal do you want to create?
```

```
Press 1 for dog,
2 for cat
3 for cow
3
Enter the name of the cow
Molly
Press 1 to view the animals,
2 to add animals,
any other key to exit
1
Hershey says Woof
Sami says Meow
Molly says Moo
Press 1 to view the animals,
2 to add animals,
any other key to exit
```

Practice Exercise

1. Add another method `setFood` to the superclass takes `String food` as parameter and sets it to the attribute `food` and returns nothing (`void`) and override it in each of the animal classes.

2. Add another method `getFood` to the superclass that returns the food as String.

3. Include the food in the `toString` method.

▶ Click here for sample code

Conclusion

In this lab, you learned how to create classes, subclasses, and use polymorphism.

Author(s)

Lavanya