Java Programming

2-3: Java Class Design – Abstract Classes

Practice Activities

Lesson Objectives:

- Use Abstract Classes
- Use virtual method invocation
- Use the instanceof operator to compare object types
- Use upward and downward casts

Vocabulary:

Identify the vocabulary word for each definition below.

Downcasting	The type of casting that changes a generalized object to a more specialized object type.
Virtual Method	The process of a call to a generalized method and actually calls the instantiated subclass method, or appropriate subclass method.
instanceof	The operator that allows you to compare a class instance against a class type.
Casting	The process of explicitly changing one data type to another data type.
Abstract Class	A class with an abstract constructor and at least one method that is defined but not implemented.
Downcasting	This type of casting changes a specialized object instance into a generalized instance. It doesn't lose any of its detail but you can't access them without downcasting the object to access specialized methods.
Abstract	A constructor without implementation that makes the class restricted in that it cannot create instances.

Try It/Solve It:

1. Update the JavaBank.java application to use the toString() methods to display the bank account details to the text area in the Java application.

   a) Update the myAccounts array definition to use the AbstractBankAccount class as its base class.

   Done

   b) Update the displayAccountDetails() method to accept a single parameter of type AbstractBankAccount named account.

   Done

   c) Call the account objects toString() method to provide the text for the JTextArea.

   Done

2. Give one reason why you might use an Abstract class rather than an Interface

   Abstract classes can have constructors, and interfaces cannot have constructors.

3. Currently in your bikeproject you can instantiate an object based on the super class Bike. Update the Bike class so that you cannot create a Bike object.

   Make a private constructor

4. Remove the bike4 code from the driver class.

   Done

5. Convert the printDescription methods in the classes to toString methods and update the code that displays the object values to the console.

   Done

6. Given the following classes.

```
public class Animal {
        public void makeNoise() {
                System.out.println("talk");
        }//end method makeNoise
}//end class Animal


public class Dog extends Animal {
        public void makeNoise() {
                System.out.println("Bark");
        }//end method makeNoise
}//end class Dog
```

a) What would the output of the following be? Explain your answer.

```
Animal animal = new Animal();          talk
animal.makeNoise();
Dog dog = new Dog();                   Bark
dog.makeNoise();
        animalDog
Animal ~~animaldog~~ = new Dog();      Bark
animalDog
~~animaldog~~.makeNoise();
```

Dog heir to the Animal class.
But create an object of class dog.

b) Using the animal and dog classes above. If we added the following code to the driver what would the output be:

```
if (animal instanceof Animal)
        System.out.println("animal is Animal");        animal is Animal
if (dog instanceof Animal)
        System.out.println("dog is Animal");           dog is Animal
    animalDog
if (~~animaldog~~ instanceof Animal)
                        animalDog
        System.out.println("~~animaldog~~ is Animal");  animalDog is Animal
if (animal instanceof Dog)
        System.out.println("animal is Dog");           The superclass knows nothing
                                                       about their descendants
```

7. Describe casting both for primitives and objects.

8. Using the animal and dog classes above. Show examples of using a downcast and an upcast.

animal instanceof Dog or animal(Dog) is Downcasting

dog instanceof Animal or dog(Animal) is Upcasting