Java Programming

3-1: Generics

Practice Activities

Lesson Objectives:

- Use enumerated types
- Create a custom generic class
- Use the type interface diamond to create an object
- Use generic methods
- Use wildcards

Vocabulary:

Identify the vocabulary word for each definition below.

Generics	This is a special type of class that associates one or more non-specified Java types.
< >	A type interface diamond is what 2 characters
Enum	A datatype that contains a fixed set of constants

Try It/Solve It:

1. For this exercise you will have to update the **JavaBank application** to include an option for the customer to choose their account type when they create an account. This will include some changes to the GUI to accommodate the new features. Follow the given instructions to update the bank application's user interface.

   a. Add a private JComboBox named accountTypes above the MAXACCOUNTS statement. Also create a private field named actType of type AccountType that will store the type of chosen account, this should be set to the default savings account.

```
//combo box to display the account types.
private JComboBox<AccountType> accountTypes;
private AccountType actType = AccountType.SAVINGS;
// constants
//public final static Maximum Accounts that can be created;
```

   b. Under the code that sets up the displayJButton but before the code that sets up the displayJLabel add the following code:

```
// set up accountTypes combo box
accountTypes = new JComboBox<AccountType>(AccountType.values());
accountTypes.setBounds(16, 238, 176, 24);
inputDetailJPanel.add(accountTypes);
accountTypes.addActionListener(
   new ActionListener() // anonymous inner class
   {
      // event handler called when TransactionJButton
      // is clicked
      public void actionPerformed(ActionEvent event) {
          //accountTypes = (JComboBox)event.getSource();
          actType = (AccountType)accountTypes.getSelectedItem();
      }//end method actionPerformed
   }//end ActionListener
); // end call to addActionListener
```

c. Run the application, click proceed when the error message is displayed! You will see that the combo box is not displayed properly as some of the interface components require to be updated.

d. Update the size of the window.

   i. Go to the end of the createUserInterface method and identify the code that sets the values for the application's window.

   ii. Update the size of the window from (670, 308) to increase the height to 340.

```
setSize(670, 340); // set window size
```

e. Update the size of the panel that contains the interactive components.

   i. Identify the code that sets up the inputDetailJPanel.

   ii. Currently the bounds settings (X, Y, Width, Height) are (16, 16, 208, 250). You need to increase the Height of the window by 30 pixels.

```
inputDetailJPanel.setBounds(16, 16, 208, 280);
```

f. Update the size of the text area to better suit the new size of the application window.

   i. Identify the code that sets up the displayJTextArea.

   ii. Increase the Height of the window to a value of 245.

```
scrollPane.setBounds(240,48,402,245);
```

2. The following task will update the required code to allow the user to select a bank account from the drop-down list.

   a. Update the myAccounts array declaration so that it instantiates AbstractBankAccount objects and not Account objects.

```
static AbstractBankAccount myAccounts[] = new AbstractBankAccount[MAXACCOUNTS];
```

   This is because you are going to add both credit and savings accounts in the application. The Array must be based on the superclass so that all of the subclasses can be stored in the array.

   b. Update the if statement that adds accounts to the system (After the finally block in the createAccountJButtonActionPerformed method) to include an if statement that checks the account type.

```
if ((noAccounts <= 9) & (name != "") & (accountNum != 0))  {
   if(actType.equals(AccountType.CREDIT)) {
      myAccounts[noAccounts] = new CreditAccount(name,accountNum,balance);
      actType = AccountType.SAVINGS;
      accountNames[noAccounts] = "USED";
   }
   else {
      myAccounts[noAccounts] = new Account(name,accountNum,balance, actType);
      accountNames[noAccounts] = "USED";

   }//endif
```

```
            displayAccountDetails(myAccounts[noAccounts]);
            noAccounts ++;
            System.out.println(noAccounts);
        }
        else {
                displayJTextArea.setText("Both the Name field and Account Number must
                                        be completed");
        }//endif
```

3. The following tasks will add functionality to the **bikeproject**.

   a. Create an Enum class named BikeUses that includes the following values:

      off_road,
      track,
      road,
      downhill,
      trail

   b. Update the interfaces to use the enum for the terrain.

      i. Update the MountainParts interface by commenting out the original line that sets the terrain and replace it with the Enum declaration using the off_road value.

```
public interface MountainParts {
        //constant declaration
        //public final String TERRAIN = "off road";
        public final BikeUses terrain = BikeUses.off_road;
```

      ii. Do the same with the RoadParts interface using the Enum value track for the terrain.

      iii. Update the toString() method in both bike subclasses to also display the terrain information so tha the output matches:

```
Oracle Cycles
This bike has Bull Horn handlebars on a Hardtail frame with 27 gears.
It has a dropper seat with Maxxis tyres.
This mountain bike is a Pro bike and has a RockShox XC32 suspension and
a frame size of 19inches.
This bike is best for off_road
```

4. You will now create a genericshapes project that will allow the use of a generic class.

   a. Create a generic class called Cuboid that will store the three dimensions of a cuboid. Add methods to set and get the length, breadth and Height. Add a method public String toString() that will return all of the dimensions. The type of the dimensions will be decided at construction of the cuboid instance. Example:

```
Cuboid<Double> c1 = new Cuboid<>();

Cuboid<Integer> c1 = new Cuboid<>();
```

   b. Create a driver class that will instantiate the Double cuboid with the values 1.3, 2.2 and 2.0.

   c. Create a driver class that will instantiate the Integer cuboid with the values 1, 2 and 3.

   d. Modify the generic class Cuboid so that it only accepts Numbers. Add a method that returns the volume of the shape as a double (l*b*h)

   e. In the driver class display the result of calling the getVolume() method on both the c1 and c2 objects.