Quiz: Purchase Class

Using the Purchase in chapter 5 of Savitch, what is the output Java code below?

```java
public static void main(String[] args) {
    Purchase orange = new Purchase();
    orange.setNumberBought(5);
    Purchase egg = orange;
    egg.setNumberBought(7);
    System.out.println(orange.getNumberBought());
    System.out.println(egg.getNumberBought());
}
```

Sets instance variable, `private int numberBought`, to 5.

a) 5              b) 7              c) 5                  d) 7
7                  5                 5                   7
Variables of a **Class** Type are *Memory Addresses*

```java
public static void main(String[] args)
{
    Purchase orange = new Purchase();
    orange.setNumberBought(5);

    Purchase egg = orange;
    egg.setNumberBought(7);

    Purchase grape = new Purchase();
    grape.setNumberBought(11);
}
```

<table>
<thead>
<tr>
<th></th>
<th>name</th>
<th>groupCount</th>
<th>groupPrice</th>
<th>numberBought</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>orange</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>egg</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>grape</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Swing Top-Level Containers

- Swing is Java's primary Application Programmer’s Interface (API) for providing a Graphical User Interface (GUI).
- Swing is part of the Java Foundation Classes (JFC).

At least one of these components must be present in every Swing application:

- **JApplet**
- **JDialog**
- **JFrame**
Swing General-Purpose Containers

(a few) Swing Controls
Coordinate System Frame of Reference

In the standard Java packages, the coordinates used in method calls are relative to the objects container. For example,

- A JFrame is a top level container. It is placed directly on the display screen. Therefore, in setBounds(left, top, width, height) the coordinates are relative to the display screen.

- A JButton might be added to an instance of a JPanel. In this case, the coordinates used in the JButton's setBounds(left, top, width, height) method are relative to the instance of its parent JPanel belongs.

Review Quiz: for Loop

Which for loop on line (1) will result in the output: 1 6 11 16

1) 2) { System.out.print(i + " ");
3) }
4) System.out.println(" ");

a) for (int i=1; i<20; i=i+5)
b) for (int i=1; i<6; i<11; i<16)
c) for (int i=1; i<16; i=i+6)
d) for (int i=1; i<=16; i=i+6)
e) for (int i=1; i<=11; i=16)
Create New Project: ToyGUI

Add one empty class source file:
- ToyGUI.java

Requirements:
1. Creates and displays three windows (JFrame) each with a different title.
2. Each of the window must be created in a different location.
3. The program must exit when any one of the windows is closed.

Extending a JFrame

```java
import javax.swing.JFrame;
public class ToyGUI extends JFrame {
    public ToyGUI(String name, int width, int height) {
        this.setTitle(name);
    }
}
```

- `this` is an instance of a GUI Frame. Since a GUI_Frame extends a JFrame, this comes with many fields and methods.

- Special method called a **constructor**. Must have the same name as the class. Has no return type (not even `void`).
ToyGUI: setBounds

```java
import javax.swing.JFrame;

class ToyGUI extends JFrame {
    public ToyGUI(String name, int width, int height) {
        this.setTitle(name);
        this.setBounds(0, 0, width, height);
    }
}
```

- GUI components, including the JFrame, JButton, JLabel, JTextField, and many others, implement a setBounds method:
  ```java
  setBounds(left, top, width, height);
  ```
- Each of these parameters are given in screen pixels.
- Left and top are relative to the component's container.
- For a JFrame, the component's container is the display screen.

ToyGUI (in progress)

```java
import javax.swing.JFrame;

class ToyGUI extends JFrame {
    public ToyGUI(String name, int width, int height) {
        this.setTitle(name);
        this.setBounds(0, 0, width, height);
        this.setVisible(true);
    }
}
```

In this version, we create the ToyGUI objects, but do not need to save references to those objects.

```java
public static void main(String[] args) {
    new ToyGUI("Ant", 200, 200);
    new ToyGUI("Bat", 300, 200);
    new ToyGUI("Cat", 400, 200);
}
```
**Run ToyGUI**

- Running ToyGUI creates and displays three, empty JFrame objects titled “Ant”, “Bat”, and “Cat”.
- Each JFrame starts at (0,0), the upper left corner of the screen.
- NOTE: The screen capture shows the windows after two of them have been moved a bit.

These windows can be resized, minimized, maximized, and moved in the way usual for that the Operating System (OS).

**Close JFrame, Program Still Running**

- Closing each of the JFrame causes the window to disappear, but does not exit the program.
- Even after closing all three windows, the program continues to run.
- If you close the three windows and then click run again, there will be two instances of the program running, each in its own Java Virtual Machine. This can be very confusing when debugging.
- Exit the program by clicking Eclipse’s Stop button:
The easiest way to cause a program to exit when the JFrame is closed is to set the default close operation to exit the Java virtual machine:

```java
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Generally, the above method call is added to the JFrame’s constructor.

Add `this.setDefaultCloseOperation` to ToyGUI’s constructor.

Run ToyGUI.

A more sophisticated control is to add window listeners to the JFrame, wherein customized behaviors can be coded when a window is closed.

---

Summery of Project: ToyGUI

```
import javax.swing.JFrame;
public class ToyGUI extends JFrame {
    public ToyGUI(String name, int width, int height) {
        public static void main(String[] args) {

            Creates three empty windows of different sizes.
            Each of the windows is placed with its upper left corner at screen pixel (0,0).
            If any one of the windows is closed, then the program exits.
            NEED TO DO: Each window must be created in a different location.
        }
    }
```

Problem: Automatic Location Change

- When ToyGUI runs, it creates three windows all with their upper left corner in the same location. If the top window is the largest, then a user might not notice that the other windows exist.

- Without adding or changing any parameters of ToyGUI’s constructor, how can the code be modified so that the first window is created with its upper left corner at (0,0), the second at (30,30), the third at (60,60), the forth at (90,90), ...?

Solution to Automatic Location Change

```java
import javax.swing.JFrame;
public class ToyGUI extends JFrame {
    private static int windowCount = 0;

    public ToyGUI(String name, int width, int height) {
        this.setTitle(name);
        int corner = windowCount*30; // Pixels
        windowCount++;
        this.setBounds(corner, corner, width, height);
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String[] args) {
    }
}
```
### ToyGUI: Adding Buttons

- **Change:** `public class ToyGUI extends JFrame` to be able to listen for button clicks:
  
  ```java
  public class ToyGUI extends JFrame
  implements ActionListener
  ```

- **Update** `ToyGUI` constructor:
  1. Get JFrame's contentPane.
  2. Create and place buttons on JFrame's contentPane.
  3. Add ActionListener to each button.

- Add `actionPerformed` method to `ToyGUI` that responds to button clicks.

### ToyGUI: Structure

```java
//imports

public class ToyGUI extends JFrame implements ActionListener
{
  //class variables

  public ToyGUI(String name, int width, int height)
  {  //Constructor: Cannot be static.
     //Set up Frame, create buttons, add listeners
  }

  public void actionPerformed(ActionEvent arg0)
  {  //Callback: Cannot be static. Respond to button clicks
  }

  public static void main(String[] args)
  {  //Must be static. Instantiate 3 instances of the GUI_Frame
  }
}
```
public class ToyGUI extends JFrame implements ActionListener
{
    ■ When JButtons are clicked, they create ActionEvents.
    ■ To respond to these events, the program must have an ActionListener.
    ■ In our program, ToyGUI will be the ActionListener.
    ■ A class that implements ActionListener must implement the method actionPerformed().
    ■ actionPerformed is a callback method. When a button is clicked, the Java Virtual Machine calls this method in the class assigned as the button’s listener.

    public void actionPerformed(ActionEvent arg0)
    { //Respond to button clicks
    }
}

import javax.swing.JFrame;  //Window frame
import javax.swing.JButton; //Buttons
// java.awt.Container is used for the For contentPane.
// The contentPane is were GUI components are placed.
// In this project, the GUI components will be buttons.
import java.awt.Container;

// When a button is clicked, it creates an ActionEvent object.
// This object contains information about the action.
import java.awt.event.ActionEvent;

// When buttons are clicked, they create ActionEvents.
// To respond to these events, the program must have an ActionListener.
// In our program, the class ToyGUI will be set up as the ActionListener.
import java.awt.event.ActionListener;
ToyGUI: Class Variables

//Class variables and methods should always be private unless
//they NEED to be accessed outside the class.

//Gets rid of serializable class warning.
private static final long serialVersionUID = 1L;

//Used to count the number of times the constructor is called.
//Need to be static because this count needs to persist beyond
//each instance of the GUI_Frame class.
private static int windowCount = 0;

//References to the buttons need to be non-static class variables
//1) Created in the constructor.
//2) Used in the actionPerformed() callback method.
//3) Each instance of ToyGUI gets its own buttons.
private JButton butOK, butCancel;

Four Steps in Creating a JButton

//Create a JButton with the label "OK".
//A reference to the JButton is stored in the variable butOK.
//butOK must have been declared as a JButton.
butOK = new JButton("OK");

//After being created, the JButton must be added to a GUI container.
//There are many types of GUI containers. In ToyGUI, the
//container used will be the ContentPane of GUI_Frame.
contentPane.add(butOK);

//Set the bounds of the JButton.
//left and top are relative to the button's container: contentPane.
butOK.setBounds(10,10,100,30); // (left, top, width, height)

//To listen to a button's events, a listener must be added to the button.
//The listener must be a class that implements ActionListener.
butOK.addActionListener(<reference to class instance>);
ToyGUI Constructor: 1 of 5

```java
public ToyGUI(String name, int width, int height)
{
    this.setTitle(name);
    int cornerPixel = windowCount*30;
    windowCount++;
    this.setBounds(cornerPixel, cornerPixel, width, height);
    this.setVisible(true);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
```

- The code shown above was copied form the first version of ToyGUI.
- In version 2, all of this code will be reused, and more code will be added to the constructor.

ToyGUI Constructor: 2 of 5

```java
public ToyGUI(String name, int width, int height)
{ //Code from version 1
    Container contentPane = this.getContentPane();
    contentPane.setLayout(null);
}
```

- GUI components cannot be placed directly on a JFrame.
- GUI components are added to the JFrame's ContentPane.
- When GUI components are added to a container, a layout manager decides how big and where the components should be.
- For simple GUIs, layout managers works fine.
- In general however, building a user friendly GUI requires human skills.
- Setting the layout manager to null turns it off.
- With the Layout manager off, the size and location of each component must be set by programmer. This will be done by using setBounds().
**ToyGUI Constructor: 3 of 5**

```java
public ToyGUI(String name, int width, int height) {
    //Code from version 1

    Container contentPane = this.getContentPane();
    contentPane.setLayout(null);

    //Create JButtons
    butOK = new JButton("OK");
    butCancel = new JButton("Cancel");

    //Add JButtons to the JFrame’s ContentPane.
    contentPane.add(butOK);
    contentPane.add(butCancel);

    //Set the size and location of the buttons (Pixels).
    butOK.setBounds(10,10,100,30);
    butCancel.setBounds(10,50,100,30);
```

**ToyGUI Constructor: 4 of 5**

```java
public ToyGUI(String name, int width, int height) {
    //Code from Version 1
    //Get contentPane and set layout manager to null.
    //Create, add, and set bounds of JButtons.
    //Add an action listener to each JButton.
    butOK.addActionListener(this);
    butCancel.addActionListener(this);
}
```

The action listener added must be an instance of a `class` that implements ActionListener. Here, `this` is an instance of Toy_GUI. Add listeners AFTER the code is ready to respond.
**ToyGUI Constructor: 5 of 5**

```java
public ToyGUI(String name, int width, int height)
{
    this.setTitle(name);
    int cornerPixel = windowCount*30;
    windowCount++;
    this.setBounds(cornerPixel, cornerPixel, width, height);
    this.setVisible(true);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    Container contentPane = this.getContentPane();
    contentPane.setLayout(null);
    butOK = new JButton("OK");
    butCancel = new JButton("Cancel");
    contentPane.add(butOK);
    contentPane.add(butCancel);
    butOK.setBounds(10,10,100,30);
    butCancel.setBounds(10,50,100,30);
    butOK.addActionListener(this);
    butCancel.addActionListener(this);
}
```

**Callback method: ActionPerformed()**

```java
public void actionPerformed(ActionEvent arg0)
{
    Object obj = arg0.getSource();

    if (obj == butOK)
    { System.out.println("Clicked OK of the " + this.getTitle() + " Frame");
    }
    else if (obj == butCancel)
    { System.out.println("Clicked Cancel of the " + this.getTitle() + " Frame");
    }
}
```
import javax.swing.JFrame;
import javax.swing.JButton;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ToyGUI extends JFrame implements ActionListener
{
    private static final long serialVersionUID = 1L;
    private static int windowCount = 0;
    private JButton butOK, butCancel;

    public ToyGUI(String name, int width, int height)
    {
        this.setTitle(name);
        int cornerPixel = windowCount * 30;
        windowCount++;
        this.setBounds(cornerPixel, cornerPixel, width, height);
        this.setVisible(true);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container contentPane = this.getContentPane();
        contentPane.setLayout(null);
        butOK = new JButton("OK");
        butCancel = new JButton("Cancel");
        contentPane.add(butOK);
        contentPane.add(butCancel);
        butOK.setBounds(10, 10, 100, 30);
        butCancel.setBounds(10, 50, 100, 30);
        butOK.addActionListener(this);
        butCancel.addActionListener(this);
    }

    public void actionPerformed(ActionEvent arg0)
    {
        Object obj = arg0.getSource();
        if (obj == butOK)
        {
            System.out.println("Clicked OK of the " + this.getTitle() + " Frame");
        } else if(obj == butCancel)
        {
            System.out.println("Clicked Cancel of the " + this.getTitle() + " Frame");
        }
    }

    public static void main(String[] args)
    {
        new ToyGUI("Ant", 200, 200);
        new ToyGUI("Bat", 300, 200);
        new ToyGUI("Cat", 400, 200);
    }
}

Quiz: Variable Cannot Be Resolved

1) import javax.swing.JFrame;
2) import javax.swing.JButton;
3) public class Test extends JFrame
4) {
5)     private JButton butOK;
6) 
7)     public Test()
8)     { this.setBounds(0, 0, 200, 300);
9)         contentPane.setLayout(null);
10)        butOK = new JButton("OK");
11)        contentPane.add(butOK);
12)        butOK.setBounds(10, 10, 100, 30);
13) }
14) 
15)     public static void main(String[] args)
16)     { new Test();
17) }
18) }

This program will not compile. It reports the syntax error "Variable Cannot Be Resolved" on line(s):

a) Lines 9 and 11
b) Line 10
c) Line 12
d) Lines 10 and 12
e) Lines 5, 10 and 12
import java.awt.Insets;
import javax.swing.JFrame;
public class Hello {
    public static void main(String[] args) {
        // No constructor. JFrame is NOT extended.
        JFrame bob = new JFrame();
        bob.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        bob.setVisible(true);
        bob.setBounds(0, 0, 200, 200); // Outside width and height

        Insets edges = bob.getInsets();
        int width = bob.getWidth() - edges.left - edges.right;
        int height = bob.getHeight() - edges.top - edges.bottom;
        System.out.println(width); // On my system: 182
        System.out.println(height); // On my system: 155
    }
}

import javax.swing.JFrame;
public class Hello {
    public static void main(String[] args) {
        JFrame fun = new JFrame();
        fun.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        fun.setVisible(true);
        for (int i = 250; i <= 600; i += 50) {
            fun.setTitle(i + "x" + i);
            fun.setBounds(0, 0, i, i);
            try {
                Thread.sleep(500); // milliseconds
            } catch (InterruptedException e) {} 
        }
    }
}
A Visual Guide to Swing Components
http://web.mit.edu/6.005/www/sp14/psets/ps4/java-6-tutorial/components.html