Introduction to Computer Science I

Graphics

Janyl Jumadinova
November 28, 2016
Abstract Classes

- **Superclasses** are created through the process called “generalization”
  - Common features (methods or variables) are factored out of object classifications (ie. classes).
  - Those features are formalized in a class. This becomes the superclass.
  - The classes from which the common features were taken become subclasses to the newly created super class.

- Often, the superclass does not have a "meaning" or does not directly relate to a "thing" in the real world.
  - It is an artifact of the generalization process.
- Because of this, abstract classes cannot be instantiated.
  - They act as placeholders for abstraction.
Abstract Classes

- **Superclasses** are created through the process called “generalization”
  - Common features (methods or variables) are factored out of object classifications (ie. classes).
  - Those features are formalized in a class. This becomes the superclass.
  - The classes from which the common features were taken become subclasses to the newly created super class.

- Often, the superclass does not have a “meaning” or does not directly relate to a “thing” in the real world.
  - It is an artifact of the generalization process.
Abstract Classes

- **Superclasses** are created through the process called “generalization”
  - Common features (methods or variables) are factored out of object classifications (i.e., classes).
  - Those features are formalized in a class. This becomes the superclass.
  - The classes from which the common features were taken become subclasses to the newly created super class.

- Often, the superclass does not have a “meaning” or does not directly relate to a “thing” in the real world.
  - It is an artifact of the generalization process.

- Because of this, abstract classes cannot be instantiated.
  - They act as place holders for abstraction.
Abstract Classes

Abstract superclass:  

`Vehicle`
- make: String
- model: String
- tireCount: int

Note: UML represents abstract classes by displaying their name in italics.

Car
- trunkCapacity: int

Truck
- bedCapacity: int
Abstract Classes

- Object-oriented programming allows classes to inherit commonly used state and behavior from other classes.
- Inheritance is declared using the “extends” keyword.
- If inheritance is not defined, the class extends a class called Object.
An interface is similar to an abstract class with the following exceptions:
- All methods defined in an interface are abstract. Interfaces can contain no implementation.
- Interfaces cannot contain instance variables. However, they can contain public static final variables (ie. constant class variables).
Interface

- Interfaces are more abstract than abstract classes.
- Interfaces are implemented by classes using the “implements” keyword.
Abstraction is especially important

- Displaying graphics is complex
- Operating system helps, but Java likes to be independent of OS

Deep hierarchy of interfaces and polymorphism (the ability of an object to take on many forms) in Java graphics packages
Swing and AWT

- AWT = Abstract Window Toolkit
  java.awt.*

- Swing is more modern
  javax.swing.*

- Every piece of a Swing GUI is a JComponent
A JFrame object represents a window

void add(Component comp)

• adds Component to window

void pack()

• automatically sizes the window around added Components

void setVisible(boolean b)

• activates the window
import javax.swing.JFrame;

public class GraphicsTester {
    public static void main(String [] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
}
JComponent Subclasses
JComponent Subclasses

```java
frame.setLayout(new FlowLayout());
frame.add(new JButton("I'm a JButton"));
frame.add(new JTextField("I'm a JTextField"));
frame.add(new JLabel("I'm a JLabel"));
frame.add(new JSlider());
frame.add(new JProgressBar());
```

FlowLayout automatically arranges left-to-right as user resizes window.
• JPanel extends Container and JComponent
• Can be used to hierarchically organize components
• add JPanels to JFrame, add JComponents to your JPanels
Painting Graphics

- All JComponent classes include a method `paint(Graphics g)`
- Swing calls `paint` on the JComponents
- Graphics2D object extends Graphics to include better OOP, rotations, etc.
- Extend JComponent to draw custom GUI elements
public class MyComponent extends JComponent
{
    /**
     * This method overrides the standard
     * JComponent paint() with our own custom code
     */
    public void paint(Graphics g)
    {
        // Custom drawing code goes here
    }
}
ActionListener Interface

- The `ActionListener` interface provides the method `actionPerformed(ActionEvent e)`
- Each GUI component keeps a collection of `ActionListener` objects
- When the user performs actions on GUI components, each `ActionListener`'s `actionPerformed()` is called
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class GraphicsTester2 {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new FlowLayout());

        JButton myButton = new JButton("I'm a JButton");
        final JTextField myTextField = new JTextField("I'm a JTextField");
        final JLabel myLabel = new JLabel("I'm a JLabel");

        myButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent event) {
                myLabel.setText(myTextField.getText());
            }
        });

        // continued in box ->

        frame.add(myButton);
        frame.add(myTextField);
        frame.add(myLabel);

        frame.pack();
        frame.setVisible(true);
    }
}
Timer

- Invisible Swing component that can call ActionListeners based on time
  - new Timer(int delay, ActionListener listener)
  - addActionListener(ActionListener listener)
  - start()
  - setRepeats(boolean b) // default true
  - setDelay(int delay) // milliseconds