Announcements

- Project 3 Due:
  - Friday, Nov. 19 - 8:00am
- Exam 2 – Wed. Nov. 10
  - Covers all material following test 1

Material

- Graphical user interfaces
  - Swing components
  - Inheritance / GUI
Redraw Architecture

- You draw to screen only in the paintComponent (or other variants) of the components/containers.
- Other parts of code set variables and force redraw.
- Division is needed because you don’t know when a redraw will take place.
Abstract Classes

• Abstract class definition:
A base class in an inheritance hierarchy containing common code, common fields of derived classes and abstract methods.

• Abstract method definition:
A method that exists in all derived subclasses, but which has no implementation in the base class. Placeholder method for derived classes for polymorphism purposes.

• Abstract classes cannot be instantiated, but abstract class reference variables can be declared to hold references to derived objects.

• All nonabstract descendants of an abstract class must over-ride abstract base class methods.
Java Abstract Classes

- Component: AWT base abstract class
- Container: AWT base abstract class
- JComponent: Swing base abstract class

Component, Container and JComponent do NOT contain abstract methods.
• Inheritance used to model “specialization”
  Base class
  Derived class (aka subclass)

• Should subclass to
  add new method(s)
  override existing method(s)

• Inheritance relationships form a hierarchy
  General class (superclass) at root
  More specific subclasses lower in tree
• Substitution Principle

Barbara Liskov

“You can use a subclass object whenever a superclass object is expected”

• Remember the Stack class in the Java API?

Is a Stack a more specialized kind of Vector?
Can you replace a Vector with a Stack anywhere?
Swing Components

Base class

Derived classes

Component
  └── Container
      └── JComponent
          ├── JPanel
          │    ├── JTextField
          │    └── JTextArea
          └── JLabel
          └── JButton
              └── JMenuItem
          └── AbstractButton
Swing components

• Base of hierarchy: Component

• Huge number of common methods:
  
  ```java
  int getWidth()
  int getHeight()
  Dimension getPreferredSize()
  void setBackground(Color c)
  ...
  ```

• Most important subclass: Container

• Example: subclass JPanel
• Typical to create subclass of JPanel

```java
public class MyPanel extends JPanel {
    
    public void paintComponent(Graphics g) {
        // drawing instructions go here
    }

    ...
}
```
Invoking Superclass

- Can’t access private fields of superclass

- To call inherited method use super
  ```java
  public void doSomething()
  {
      // code here?
      super.doSomething();
      // code here?
  }
  ```

- `super` is a special keyword that begins method dispatch from “super” class
Invoking Super Constructors

- Use `super()` in constructor
- Must be first statement in subclass constructor
- If omitted, compiler generates call to super default constructor, (i.e. no parameters)

```java
public class MyPanel extends JPanel {
    public MyPanel() {
        // if no call to super, then JPanel() is called
        super(new BorderLayout());
        ...
    }
}
```
Preconditions

- Precondition of redefined method at most *as strong*

```java
class Employee {
    /**
     * Sets the employee salary to a given value.
     * @param aSalary the new salary
     * @precondition aSalary > 0
     */
    public void setSalary(double aSalary) { ... }
}
```

- Can we redefine Manager.setSalary with precondition salary > 100000?
- No--consider this...

```java
Employee e = ... //what if e refers to a Manager object
e.setSalary(50000);
```
**Precondition “as strong”**

- Precondition is a boolean expression over some range of values. Condition divides range into “zones”

- Consider precondition: salary > 0

- Is this condition stronger or weaker: salary > 100,000
Inheritance or Interface

• Differences?
  Class can implement *many* interfaces
  Class can inherit only from *one* superclass
  Java is a single inheritance language
  C++ supports multiple inheritance

• Similarities
  Both are techniques that support abstraction, and reuse
public class MyPanel
  extends JPanel
  implements MouseListener {

  public MyPanel() {
    super(new BorderLayout());

    addMouseListener(_________);
  }

  public void mouseClicked(MouseEvent e) {
    // do something
  }

  // other MouseListener methods
}
Mixing it all

• Encapsulation
  LinesPanel as a stand alone class
  More reuseable

• But separation of data and code is good at times
  Can we separate the vector of objects from the method?
  If we do, we can have two panels with the same data, 2 views
  How to refresh the “other”?