Inheritance
Inheritance

• One class inherits from another if it describes a specialized subset of objects

• Terminology:
  – the class that \textit{inherits} is called a \textit{child class} or \textit{subclass}
  – the class that is \textit{inherited from} is called a \textit{parent class} or \textit{superclass}
Characteristics of Inheritance

• Child classes are both extensions and contractions of parent classes:
  – extensions because they can add characteristics beyond those inherited, as well as override inherited characteristics
  – contractions because they are more specialized, and thus more restricted than parent class

• Inheritance is always transitive: a class can inherit features from a superclass that has inherited from another superclass, etc.
Forms of Inheritance: Specification

- Inheritance used to guarantee that classes share a common interface
- Subclass is realization of incomplete abstract specification, not refinement of existing type
- Mechanisms for this type of inheritance include interfaces and abstract classes and methods
Forms of Inheritance: Specialization

• Most common use of inheritance
• New class is specialized variety of parent
• Satisfies specifications of parent, but extends capabilities:
  – One or more inherited methods may be overridden
  – New fields/methods may be introduced
Inheritance in Java

• Subclass denoted by keyword `extends`  
• Subclass declaration only contains differences from superclass  
  – additional fields/methods  
  – overridden methods
Inheritance hierarchies

• Real world: Hierarchies describe general/specific relationships
  – General concept at root of tree
  – More specific concepts are children

• Programming: Inheritance hierarchy
  – General superclass at root of tree
  – More specific subclasses are children
Designing class hierarchy

• Collect common properties in superclasses at base of hierarchy (root)
• Further down the tree get more specialization; classes at leaves are most specialized
Substitutability

- Idea that type given in variable declaration does not have to match type associated with value variable is holding
- Can occur through inheritance: for example, variable of superclass type can hold subclass objects
- Can also occur through use of interfaces - specifically with parameter passing
Substitutability

• Liskov substitution principle: since subclass inherits superclass behavior, can substitute subclass object when superclass object expected

• Polymorphism: client calls method on object (perceived to be) of parent type; if child object used instead, overriding method is executed
Invoking superclass methods

• Can call inherited (parent) method from overriding (child) method using keyword `super`:

```java
public void aMethod()
{
    super.aMethod(); // calls parent version
}
```
Notes on use of super

• Without keyword, method call in previous example would be recursive

• super is not a variable - doesn’t hold a reference to a superclass object - more like invoking a static method

• Turns off polymorphic call mechanism, frees superclass method to be called
Invoking superclass constructors

• Use super keyword in subclass constructor:
  
  ```java
  public Manager(String aName)
  {
      super(aName);
      // calls superclass constructor
      bonus = 0;
  }
  ```

• Here, super is used as name of method
Invoking superclass constructors

• Call to super must be *first* statement in subclass constructor

• If subclass constructor doesn't call super, superclass must have constructor without parameters
Preconditions & inherited methods

- Subclass method cannot require stronger precondition than method it overrides
- If superclass method had no precondition, subclass method can’t require one
Postconditions & inherited methods

- Subclass method:
  - postcondition must be at least as strong as postcondition of original
  - cannot be more private than original
  - cannot throw more checked exceptions than original
Graphics programming with inheritance

• To draw shapes, can implement Icon interface, as we have seen

• More versatile method: subclass Jpanel, override paintComponent() method

• Advantage: inherit rich behavior set
  – For example, can attach mouse listener to panel
  – With interface, you start from scratch; with inheritance, you get readymade features of superclass
Overriding paintComponent()

Example - draw a car:
public class MyPanel extends JPanel
{
    public void paintComponent(Graphics g)
    {
        Graphics2D g2 = (Graphics2D)g;
        car.draw(g2);
    }
    ...
}
Problem: corrupted panel

- Screen corrupted when moving pane
- Solution: call `super.paintComponent(g)` as first line in overriding method to refresh panel each time image is redrawn
Java Event Model

- Event: an action, e.g. mouse click, key press, menu selection
- Listener: object whose purpose is to wait for an event to occur, and perform appropriate action when event is detected
- Buttons & scrollbars are examples of objects that must incorporate listeners
Mouse Listeners

• There are 5 different mouse-related events, so the MouseListener interface accounts for all:

```java
public interface MouseListener {
    public void mouseClicked (MouseEvent e);
    public void mouseEntered (MouseEvent e);
    public void mouseExited (MouseEvent e);
    public void mousePressed (MouseEvent e);
    public void mouseReleased (MouseEvent e);
}
```
Mouse Listeners

• If a class implements this interface, it must provide definitions for all of these operations (whether you need them or not)
• If all operations aren’t needed, can use class MouseAdaptor instead
  – Implements MouseListener methods as empty method bodies
  – Can inherit from MouseAdaptor, then just override the methods you actually want to use
Mouse Listeners

• Alternatively, you can implement the MouseListener interface and create your own empty methods for operations you don’t need - for example:
  
  ```java
  public void mouseExited(MouseEvent e) {}
  ```

• Method addMouseListener is inherited from Frame - takes a MouseListener instance as its object
Using anonymous class to add mouse listener

addMouseListener(new MouseAdapter(){
    public void mousePressed (MouseEvent event)
    {
        mouse action goes here
    }
});