Object Oriented Programming and Design in Java

Session 10
Instructor: Bert Huang

Announcements

- Homework 2 due Mar. 3rd, 11 AM
  - two days
- Midterm review Monday, Mar. 8th
- Midterm exam Wednesday, Mar. 10th

Review

- More LayoutManager examples
  - BorderLayout, BoxLayout, GridLayout
- Discussion of Inheritance
  - Liskov's Substitution Principle
  - Polymorphism, encapsulation, preconditions and postconditions

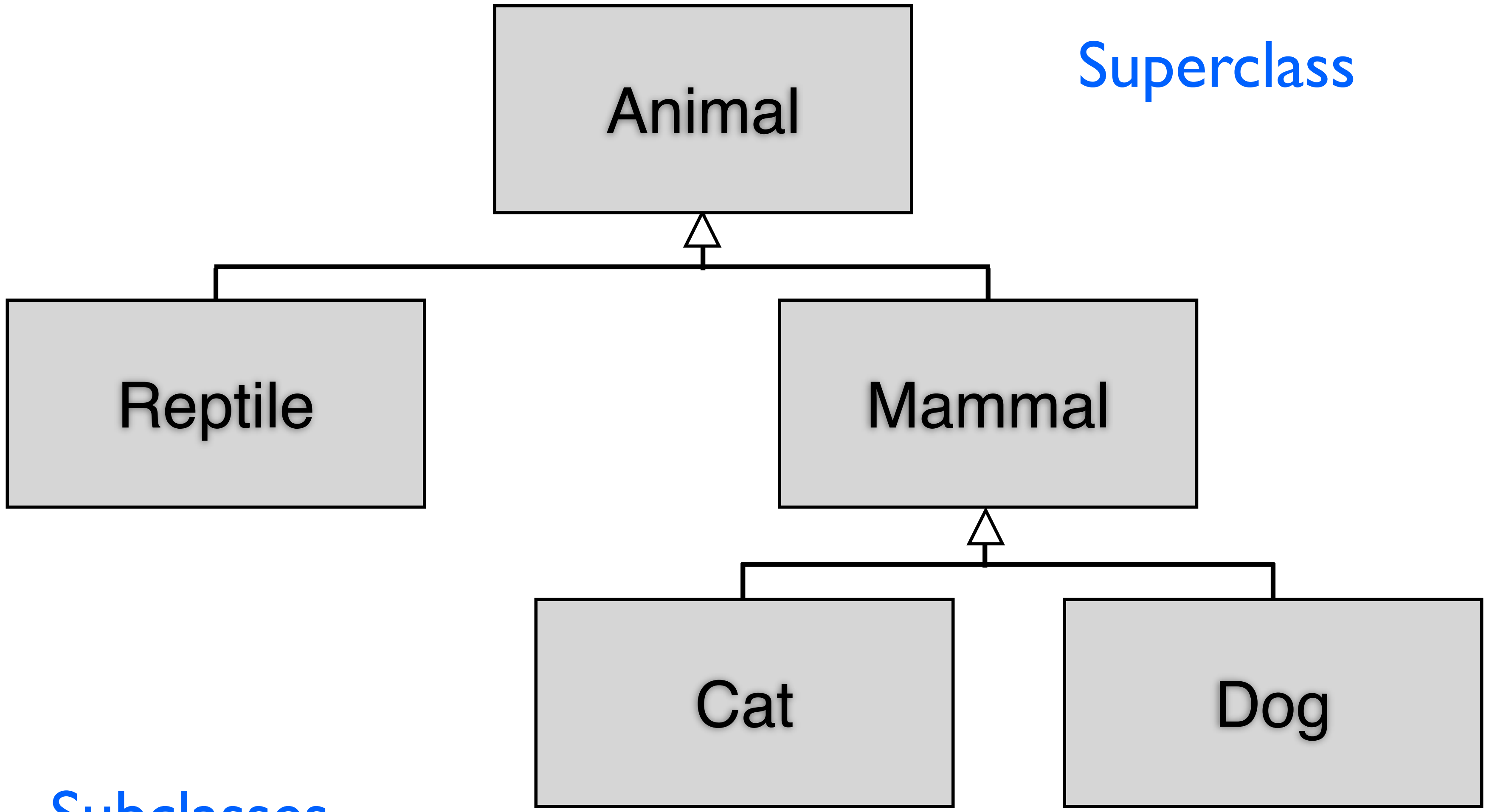
Today's Plan

- Inheritance and hierarchy
- Abstract classes
- Example hierarchies
  - Swing class hierarchy
  - awt.geom hierarchy
  - Exception hierarchy

Inheritance

- Subclasses inherit methods and fields from superclasses
- Analogous to taxonomies
- In Java and most languages, subclasses can only inherit from one superclass

Phylogenetic Trees



Superclass

Subclasses

Abstract Classes

- Abstract classes are meant to be extended by various subclasses
- The abstract class can never be instantiated
- but methods and fields can be defined and implemented
- A subclass can only extend one abstract class

Abstract Class Example

- Suppose you make a HumanPlayer and ComputerPlayer class for a card game
- CRC cards for both include
  - next move given game state
  - store score, cards
  - remember previous moves

implementation will be the same

AbstractPlayer

```
/**
 * Example class. Will not compile and features
 * a very incomplete design
 */
public abstract class AbstractPlayer {

   public AbstractPlayer()
   {
      myCards = new ArrayList<Card>();
      score = 0;
   }

   public abstract Move nextMove(GameState game);

   public void addCard(Card c) { myCards.add(c); }

   public int getScore() {  return score; }

   public void setScore(int newScore) { score = newScore; }
 */
public abstract class AbstractPlayer {

   public AbstractPlayer()
   {
      myCards = new ArrayList<Card>();
      score = 0;
   }

   public abstract Move nextMove(GameState game);

   public void addCard(Card c) { myCards.add(c); }

   public int getScore() {  return score; }

   public void setScore(int newScore) { score = newScore; }

   public void addMove(Move newMove) { myMoves.add(newMove); }

   private score;
   private ArrayList<Card> myCards;
   private ArrayList<Move> myMoves;
}
```

Template Methods

- Not always obvious how to separate algorithms and implementations
- Sometimes parts of algorithms are implementation specific, but the main flow is the same
- Think of the main flow of the algorithm as a template

Saving a file

- Format-free template method:
  - Open a file to be written
  - Translate object to be saved to text or binary format
  - Write text or binary to file
  - Close file

Pattern: Template Method

- An algorithm is applicable for multiple types
- The algorithm can be broken down into *primitive operations*. The primitive operations can be different for each type
- The order of the primitive operations in the algorithm doesn't depend on the type

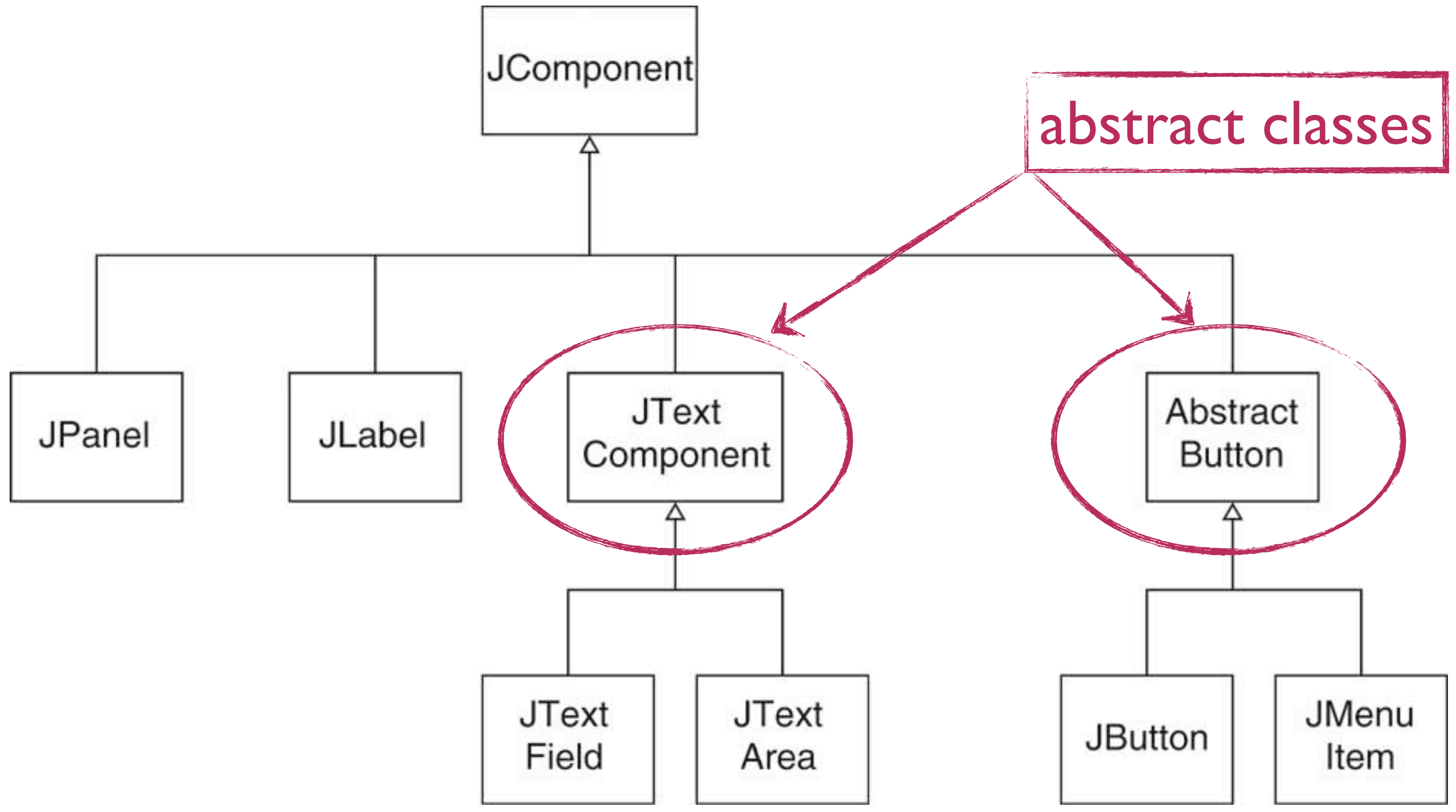
---

- Define an abstract superclass that has a method for the algorithm and abstract methods for the primitive algorithms
- Implement algorithm to call primitive operations in order
- Leave primitive operations abstract or have basic default
- Each subclass defines primitive operations but not the algorithm

Template vs. Strategy

- Template Method is very similar to Strategy
- Strategy delegates entire algorithm to the strategy object
- Template method delegates small pieces: the primitive operations

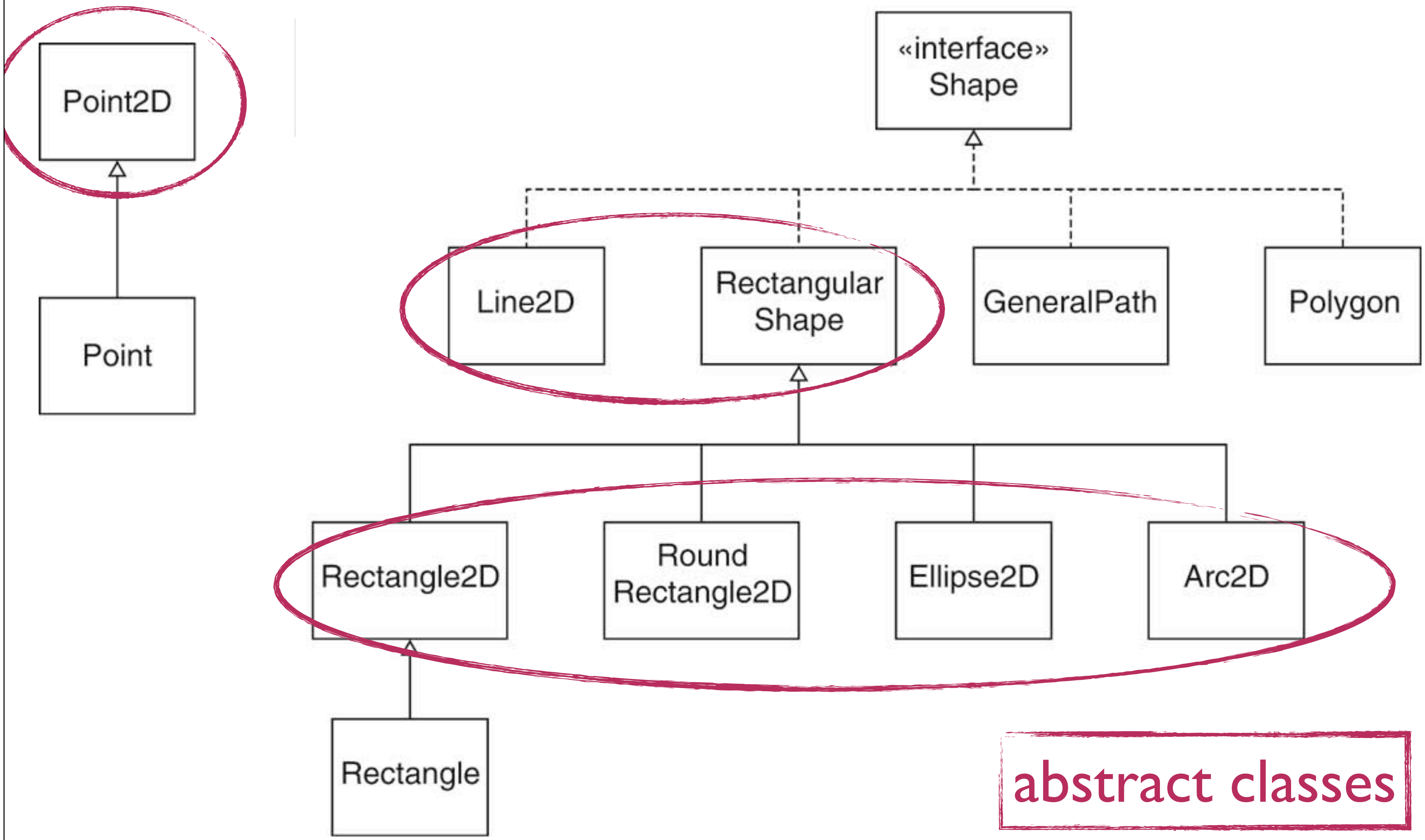
Swing Components



JTextComponent

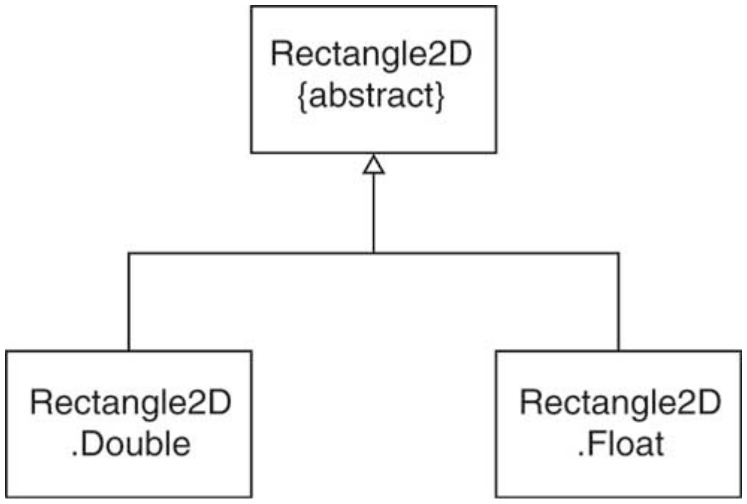
- int getSelectionEnd()
- int getSelectionStart()
- String getText()
- void setText()
- void paste()
- void setEditable(boolean)
- boolean isEditable()

AWT Shapes



Point2D
Point
«interface» Shape
Line2D
Rectangular Shape
GeneralPath
Polygon
Rectangle2D
Round Rectangle2D
Ellipse2D
Arc2D
Rectangle
abstract classes

Rectangle2D



- Rectangle2D has two inner classes
- Let's clients choose tradeoff between precision and memory
- Most work is done inside Rectangle2D (using double precision!)

```
public class Rectangle2D
{
   public static class Float extends Rectangle2D
   {
      public double getX() { return x; }
      public double getY() { return y; }
      public double getWidth() { return width; }
      public double getHeight() { return height;}
      // ...
      public float x;
      public float y;
      public float width;
      public float height;
   }

   public static class Double extends Rectangle2D
   {
      public double getX() { return x; }
      public double getY() { return y; }
      public double getWidth() { return width; }
      public double getHeight() { return height;}
      // ...
      public double x;
```

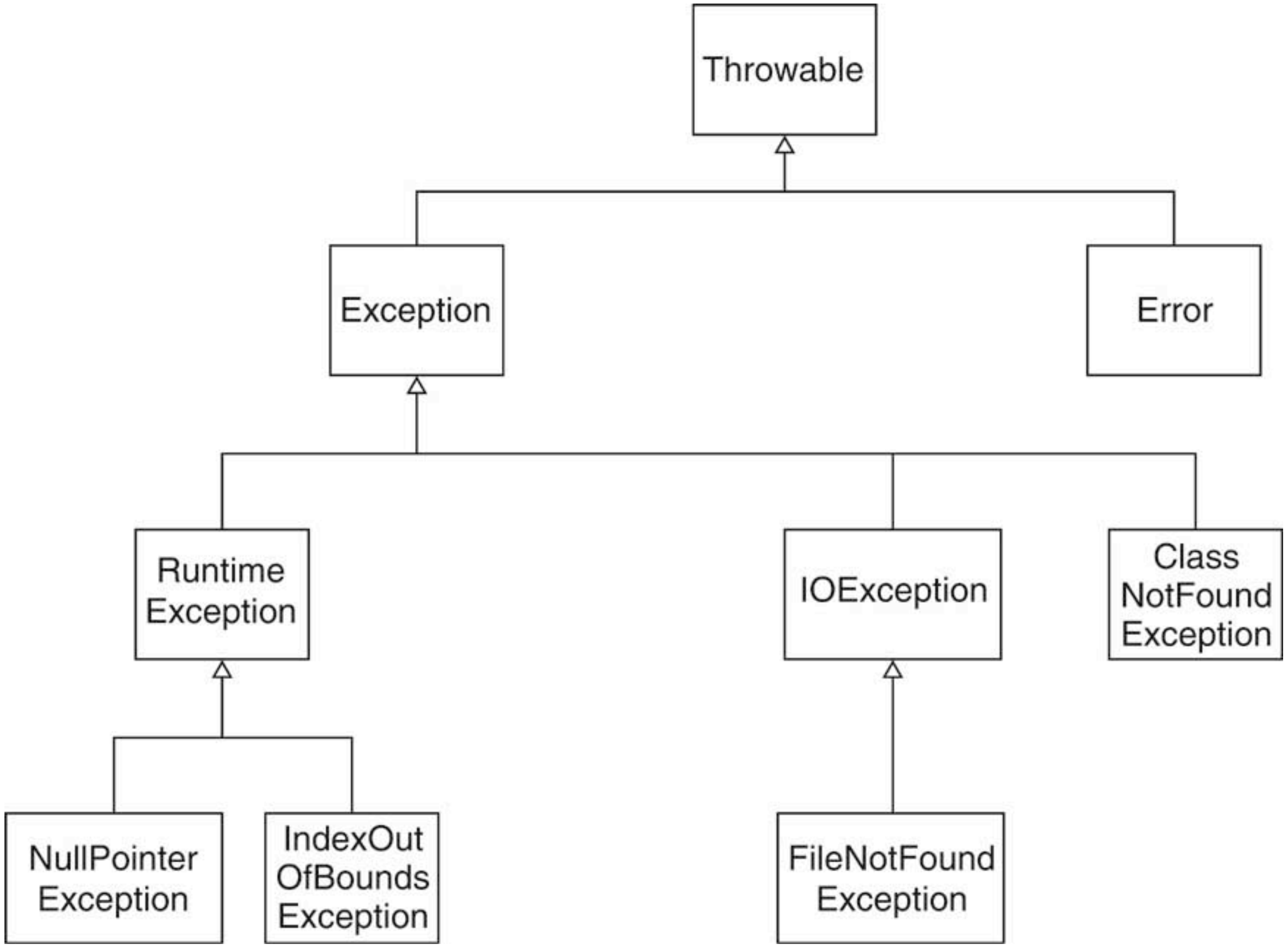
```java
  public static class Double extends Rectangle2D
  {
    public double getX() { return x; }
    public double getY() { return y; }
    public double getWidth() { return width; }
    public double getHeight() { return height;}
    // ...
    public double x;
    public double y;
    public double width;
    public double height;
  }

  public boolean contains(double x, double y)
  {
     double x0 = getX();
     double y0 = getY();
     return x >= x0 && y >= y0 &&
         x < x0 + getWidth() &&
         y < y0 + getHeight();
  }
  // ...
}
```

primitive operations

Template Method

Exceptions


Throwable
Exception
Error
Runtime Exception
IOException
Class NotFound Exception
NullPointer Exception
IndexOut OfBounds Exception
FileNotFound Exception

Hierarchy

- With some foresight, you can design inheritance hierarchy for classes
- Otherwise, when you find redundant functionality, refactor into hierarchy after or during coding

Reading

- Today:
  - Horstmann Ch. 6
- Wednesday:
  - Horstmann Ch. 7.1-7.6