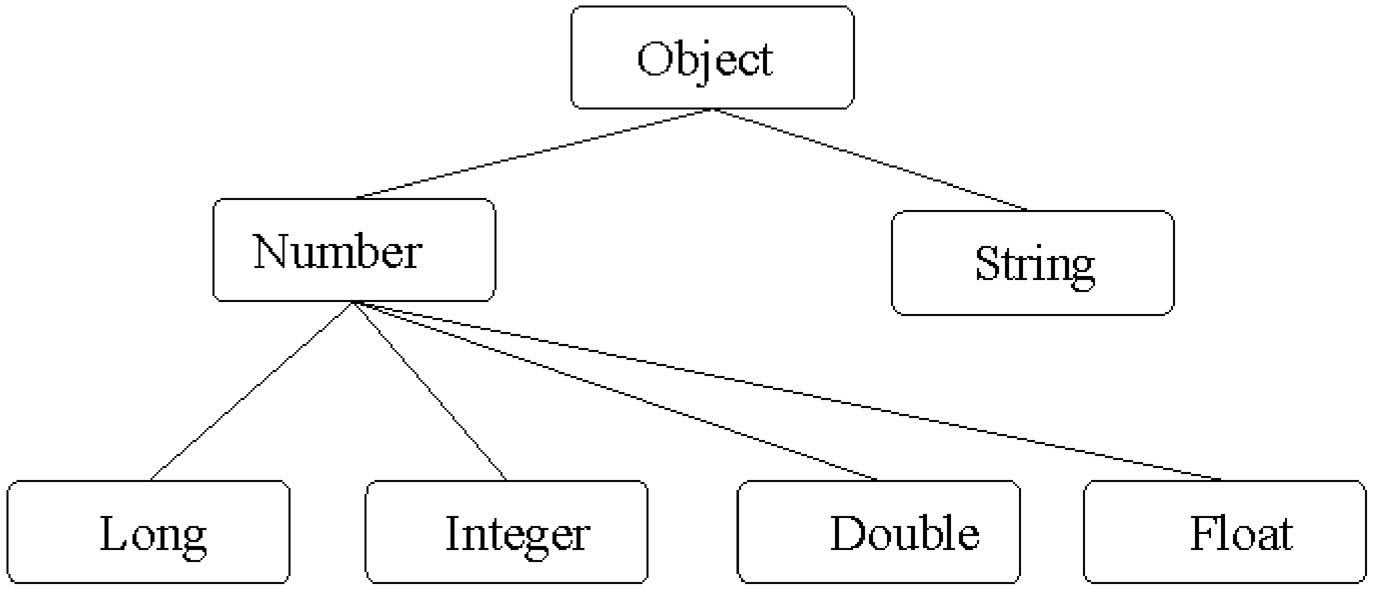
OMET

Wrapper Classes

- classes that wrap up primitive values in classes that offer utility methods to manipulate the values
- they also offer utility methods for converting to and from the int values they represent
- once assigned a value, the value of a wrapper class cannot be changed

Wrapper Classes

An example from java.lang



Wrapper Classes

```
book p. 247:

static Number elementMin(Number[] array) {
        Number min = array[0];
        for (int i=1; i<array.length; i++)
           if (array[i].doubleValue <
             min.doubleValue)
              min = array[i];
        return min;
        }
```

test yourself: http://www.jchq.net/certkey/0803certkey.htm

Objects

- primitive vs. reference type
- static vs. instance
- abstract vs. concrete vs. interface

The static keyword

- Java methods and variables can be declared static
- These exist **independent of any object**
- This means that a Class's
  - static methods can be called even if no objects of that class have been created and
  - static data is "shared" by all instances (i.e., one value per class instead of one per instance

```
class StaticTest {static int i = 47;}
StaticTest st1 = new StaticTest();
StaticTest st2 = new StaticTest();
// st1.i == st2.I == 47
StaticTest.i++;          // or st1.I++ or st2.I++
// st1.i == st2.I == 48
```

http://www.csee.umbc.edu/~nicholas/331/webFa01/lectures/

Static vs. instance

```
import java.util.Date;
class DateApp {
    public static void main(String args[]) {
        Date today = new Date();
        System.out.println(today);
    }
}
```

Abstract classes and methods

- Abstract vs. concrete classes
- Abstract classes can not be instantiated

```
public abstract class shape { }
```

- An abstract method is a method w/o a body

```
public abstract double area();
```

- (Only) Abstract classes can have abstract methods
- In fact, any class with an abstract method is automatically an abstract class

Example

abstract class

```
public abstract class Shape {
  public abstract double area();  // Abstract methods: note
  public abstract double circumference();// semicolon instead of body.
}

class Circle extends Shape {
  public static final double PI = 3.14159265358979323846;
  protected double r;                              // Instance data
  public Circle(double r) { this.r = r; }          // Constructor
  public double getRadius() { return r; }          // Accessor
  public double area() { return PI*r*r; }          // Implementations of
  public double circumference() { return 2*PI*r; } // abstract methods.
}

class Rectangle extends Shape {
  protected double w, h;                               // Instance data
  public Rectangle(double w, double h) {               // Constructor
    this.w = w;  this.h = h;
  }
  public double getWidth() { return w; }               // Accessor method
  public double getHeight() { return h; }              // Another accessor
  public double area() { return w*h; }                 // Implementations of
  public double circumference() { return 2*(w + h); }  // abstract methods.
}
```