Session 2

Outline

- Difference between applications and applets
- The first Java Program
- Inner classes,
- Abstract classes

- Wrapper Classes

Application vs. Applet

Application vs. Applet

- An application is a standalone Java program that runs as a true application, outside of a browser. Both require a JVM (Java Virtual Machine).

- An applet is a program written in the Java programming language that can be included in a HTML page, much in the same way an image is included.

- Applets are always embedded in a HTML page.

- Applications are invoked by the Java interpreter, and applets are invoked by the Web browser.

Security Restrictions on Applets

- Applets have security restrictions:
  - ✘ An applet can't touch the local disk.
  - ✘ Applets are not allowed to read from, or write to, the file system of the computer viewing the applets.
  - ✘ Applets are not allowed to run any programs on the browser's computer.
  - ✘ Applets are not allowed to establish connections between the user's computer and another computer except with the server where the applets are stored.

✓ An applet can communicate with only the originating server.

First Applet Program

```
// A first applet in Java
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorldApplet extends Applet {
        public void paint( Graphics g )
        {
        g.drawString( "Hello World!", 60,100 );
        }
}
```

No **main** in applets: **paint**() method is called by JDK's *appletviewer* or the browser2

Creating Applet

- Standard Applets are built on the Applet class, which is in the java.applet package.
- So, start by importing that class using

  import java.applet.Applet;
- The java.applet.Applet class is the class that forms the base for standard applets, and you can derive your own applet classes from this class using extends keyword:

```
import java.applet.Applet;
    public class HelloWorldApplet extends Applet{
      …..
    }
```

Creating Applet (cont'd)

- Applets don't have a main method like applications do.
- The actual drawing of an applet is accomplished in its paint method, which the JVM calls when it's time to display the applet.
- The java.applet.Applet class has its own paint method, but we can override that method by defining our own paint method, like this:

HTML Tags for An Applet

```
<html>
<title>APPLET</title>
 <applet code="HelloWorldApplet.class" width=300 height=150>
 </applet>
 </html>
```

Running An Applet

- javac HelloWorldApplet.java

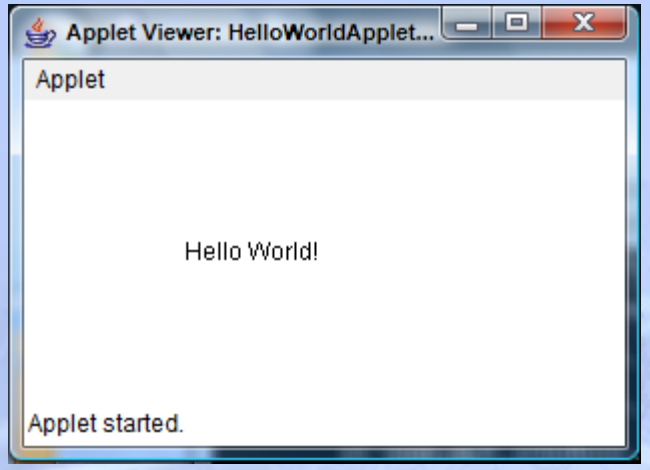
- To display an applet, a Web page with an HTML<APPLET> tag in it is used. Create HelloWorldApplet.html in same directory with HelloWorldApplet.java

  – Open this applet Web page in a Web browser

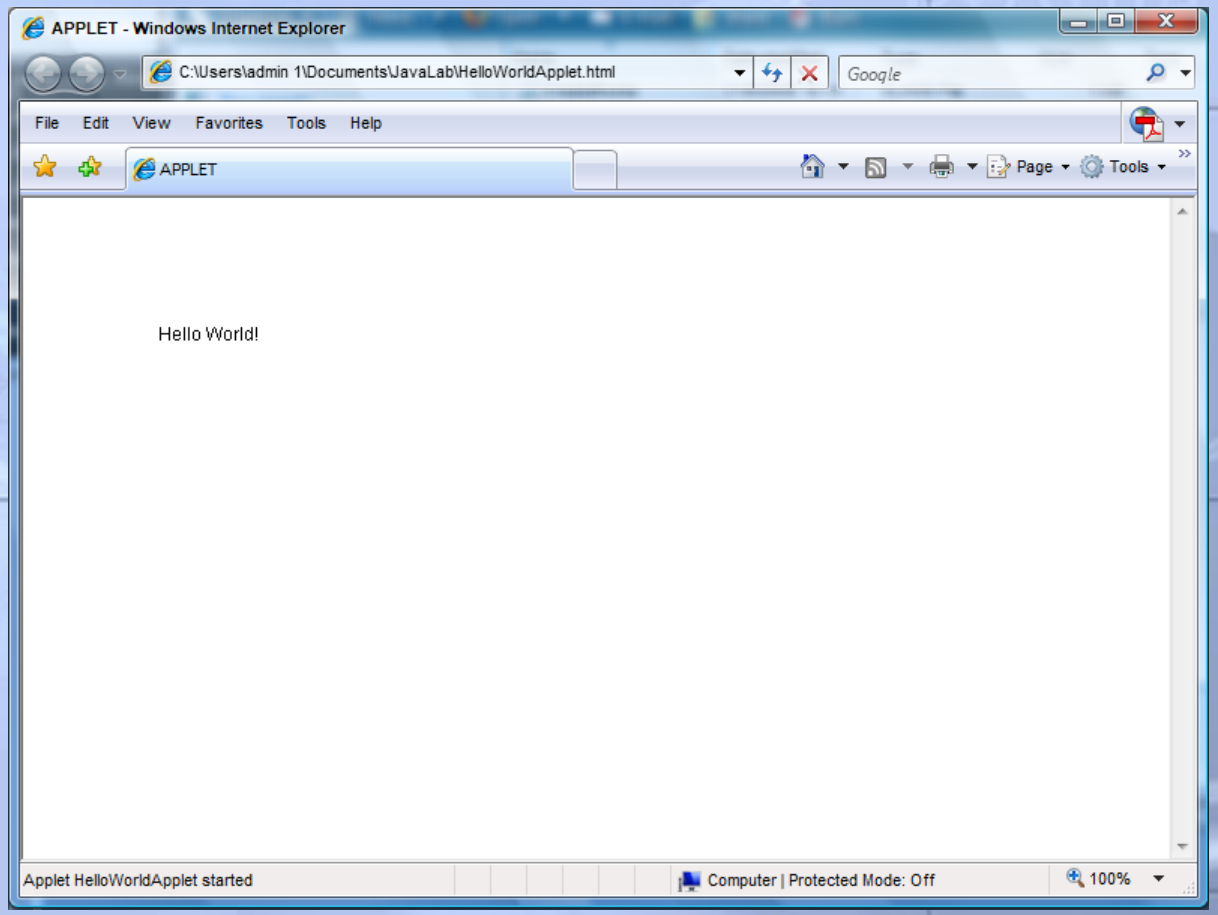
  (or)

  – Use Sun appletviewer
    - appletviewer HelloWorldApplet.html

Outputs

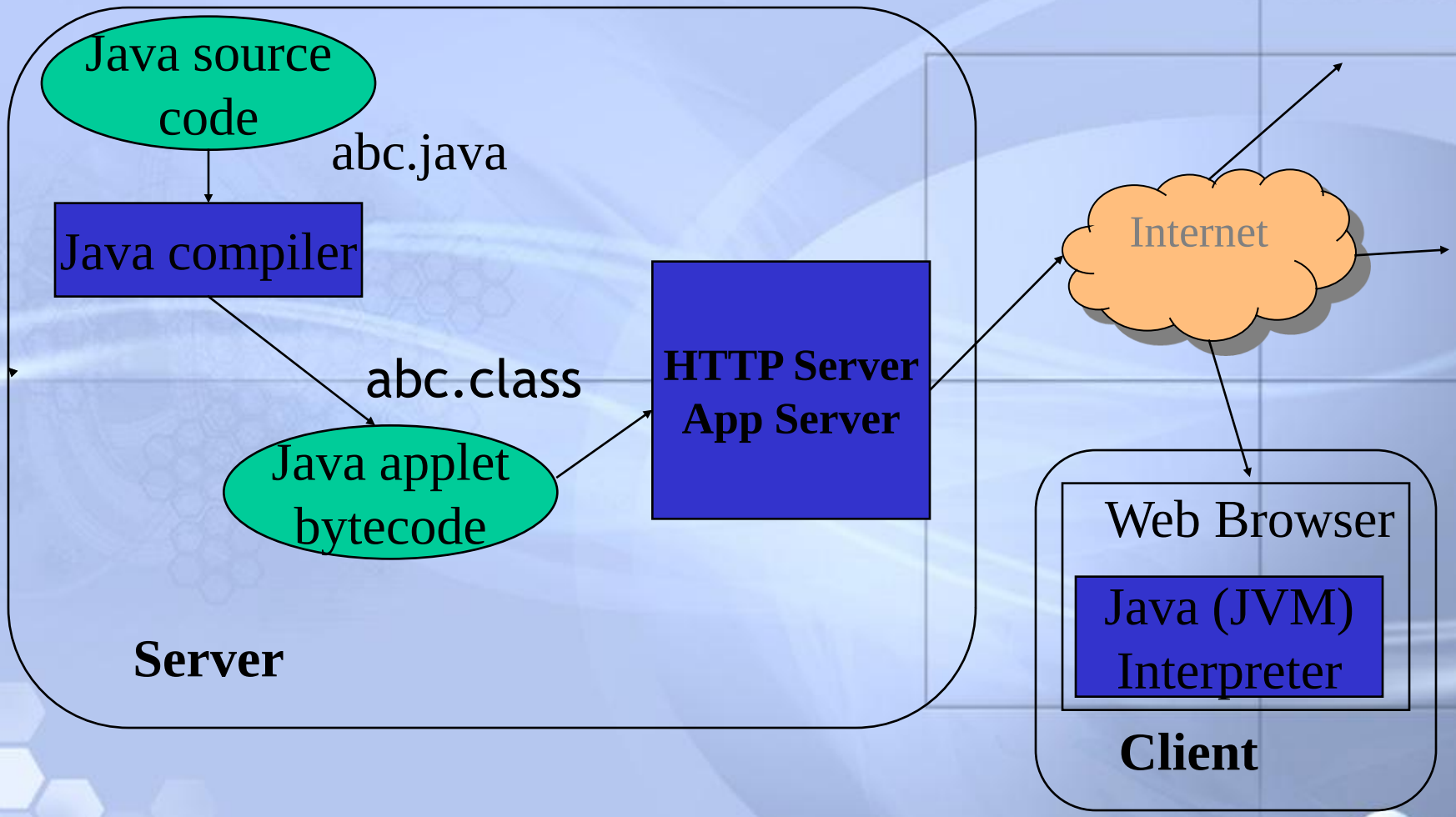


A simple Applet run in the appletviewer



The same applet run in a browser

Java Translation and Execution



First Java Program

Hello World

```
// HelloWorld.java: Hello World program
import java.lang.*;
public class HelloWorld
{
     // Print a greeting
    public static void main(String args[])
    {
       System.out.println("Hello World");
    }
}
```

Closer Look at - Hello World

- The class has one method – main()

```
public static void main(String args[])
{
    System.out.println("Hello World!");
}
```

- Command line input arguments are passed in the String array args[]

  e.g java HelloWorld John Jane

  args[0] – John args[1] – Jane

Java imports java.lang.* by default

- So, You don't need to import java.lang.*

- That means, you can invoke services of java's "lang" package classes/entities, you don't need to use fully qualified names.
  - We used

System.out.println() instead of

java.lang.System.out.println()

public static void main(String args[])

- public: The keyword “public” is an access specifier that declares the main method as unprotected.
- static: It says this method belongs to the entire class and NOT a part of any objects of class. The main must always be declared static since the interpreter uses this before any objects are created.
- void: The type modifier that states that main does not return any value.
- A program must include a *method* called **main** where the program starts. The argument to main must always be a string array (containing any command line arguments).

System.out.println("Hello World");

- java.lang.*
  - All classes/items in "lang" package of java package.

- System is really the java.lang.System class.

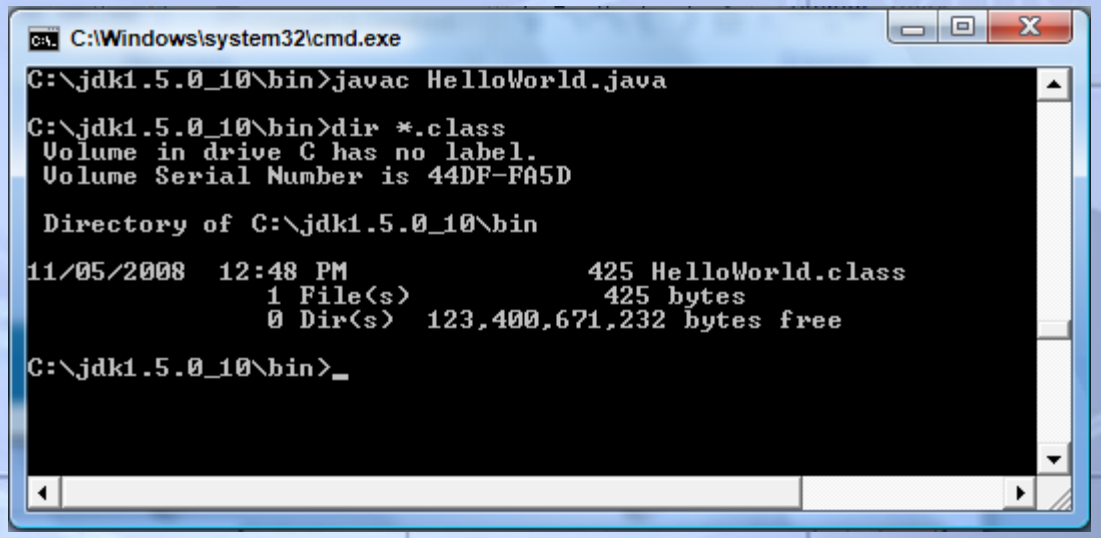
- This class has a public static field called out which is an instance of the java.io.PrintStream class. So when we write System.out.println(), we are really invoking the println() method of the "out" field of the java.lang.System class.

Compiling & Running Code

- Compilation

  # **javac HelloWorld.java**

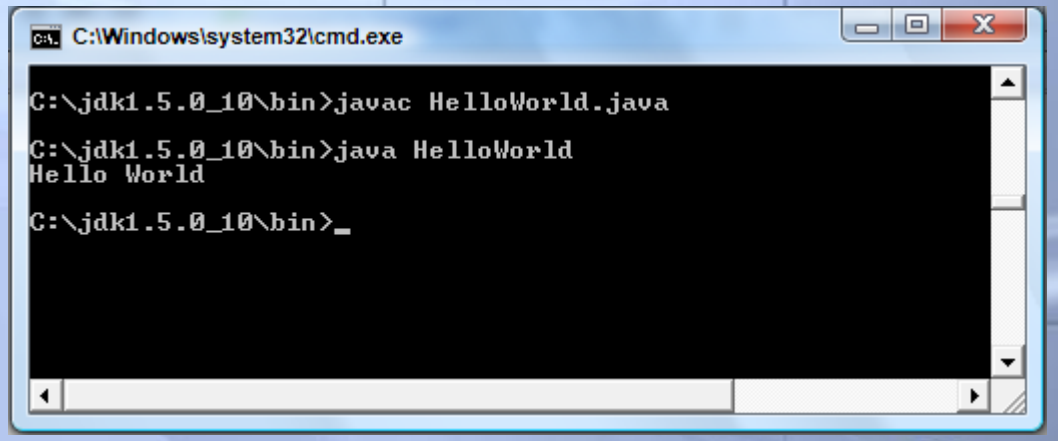
  results in HelloWorld.class



- Execution

  # **java HelloWorld**

  Hello World



Inner Classes

Inner Class

- A class which is a member of another class.

  (Declaration of a class within another class)

- An inner class requires an instance of an outer class.

- An inner class can have public, protected, default, or private access
  - just like any other member

- Inner classes can be either named or anonymous.

Inner Class (cont'd)

- A inner class defined **within a method or statement block** is *automatically* **private**.

- Code in the inner class has access to all members of the outer class
  - even private members
- Code in the outer class has access to all members of the inner class
  - even private members

- Inner class can access all final variables of enclosing method

- Inner class cannot have static variables
  - They can have static final variables i.e. compile-time constants.

Types of Inner Class

- There are four categories of inner classes in Java:

1. Inner classes (non-static).
2. Local inner classes (defined inside a block of Java code).
3. Anonymous inner classes (defined inside a block of Java code).
4. Static inner classes.

Inner Class (non-static)

- **Non-static Inner Classes**
- An inner class is declared as a member of an enclosing class. An inner class is declared like a top-level inner class except for a **static** modifier. Inner classes do not have **static** modifiers.

```
class Outer { // Top-level class

        class Inner {// without static modifier
                // Member or Non-static inner class
        }

}
```

Inner Class Example

```
public class Customer {
    private String name;
    private Address homeAddress, workAddress;
    public class Address {
        private int number;
        private String street;
        public Address(int no, String street) {
            number = no;
            this.street = street;
        }
        public String toString() {
        return "the address of " + name + " is " + number + " " + street;
        }
    } // end of inner class
    public Customer(String name, int houseNumber, String homeStreet) {
        this.name = name;
        homeAddress = new Address(houseNumber, homeStreet);
        }
}// end of outer class
```

Using Inner class

- Objects of the inner class can exist only within the context of an object of the outer class
- Within the outer class, can create inner class objects as normal

```
public class Customer {
....
    public Customer(String name, int houseNumber, String homeStreet) {
    this.name = name;
    homeAddress = new Address(houseNumber, homeStreet);
}
```

Creating Inner Class object outside Outer Class

- If the inner class is public, you can also create objects from it outside the outer class
  - you must first create an instance of the outer class for the inner class instance to belong to
  - the type of the inner class object is OuterClass.InnerClass

```
public static void main(String[] args)
{
    Customer c1 = new Customer("Cathy", 10, "Unicorn lane");
    Customer.Address  anotherAddress = c1.new Address(8, "Octagon Road");
    // See the use of new
    System.out.println(anotherAddress.toString());
}
```

Inner Class Example

```
class Outer {
        int outer_x = 100;
        void test() {
                Inner inner = new Inner();
                inner.display();
        }
        class Inner {
          int y = 10;
          void display () {
                System.out.println("display:" +" outer_x = " + outer_x);
           }
        }
}
```

- Javac Outer.java
- You'll end up with two class files
- Outer.class and Outer$Inner.class

Accessing Outer Scope

```
public class OC {                  // outer class
    int x = 2;

   public class IC {               // inner class
      int x = 6;
      public void getX() {         // inner class method
                              int x = 8;
                               System.out.println( x );          // prints 8
                              System.out.println( this.x );      // prints 6
                              System.out.println( OC.this.x );   // prints 2
      }
   }
}
```

Local Inner Class

- An inner class is **declared within the body of a method**. Such a class is known as a *local inner class*.

- The name of a local class is visible and usable only within the block of code in which it is defined *(and blocks nested within that block).*

- The methods of a local class can use any ***final*** local variables or method parameters that are visible from the scope in which the local class is defined.

Anonymous Inner Class

- An inner class is **declared within the body of a method** without naming it. These classes are known as *anonymous inner classes*.

- Anonymous class as an argument to method.
- Instance of class returned by method

- Inner class defined at the place where you create an instance of it (in the middle of a method)
  - Useful if the only thing you want to do with an inner class is create instances of it in one location
- In addition to referring to fields/methods of the outer class, can refer to final local variables

Syntax for Anonymous Inner Class

```
new ReturnType() { // unnamed inner class
body of class… // implementing ReturnType
} ;
```

Example

```
public class MyList {
    public Iterator iterator() {
        return new Iterator() { // unnamed inner class
        …                       // implementing Iterator
        } ;
    }
}
MyList m = new MyList();
Iterator it = m.iterator();
```

Simple Anonymous Class Example

```
public class MainClass {
      public static void main(String[] args) {
        Ball b = new Ball() {
                                public void hit() {
                                      System.out.println("You hit it!");
                                }
                    };
        b.hit();
      }

      interface Ball
    {    void hit();  }
}
```

Static Inner Class (nested class)

- By the standard definition of an inner class, the static inner class is not an inner class at all.
- A static nested class is simply a class that's a static member of the enclosing class

```
class Outer{
static class Nested{ }
}
```

- The static modifier in this case says that the nested class is a static member of the outer class.
- That means it can be accessed, as with other static members, without having an instance of the outer class.
- If we do not need the link between the inner class and outer class then,
  - Inner class is given **static** keyword.
  - This is commonly called as **nested class.**

Abstract Class

Abstract Class

- A superclass that only defines a generalized form that will be shared by all its subclasses, leaving the implementation details to its subclasses is said to an abstract class.
- An *Abstract* class is a conceptual class and cannot be **instantiated** – objects cannot be created.

- Keyword abstract used to create abstract class
- Abstract classes can contain whatever an "ordinary" class can:
  - instance and class variables, instance and class methods, with whatever modifiers
- Moreover, abstract classes can contain **abstract methods**
- An abstract method is given the signature only, not equipped with a body, i.e. no implementation is given for it.

Abstract Class Syntax

```
abstract class ClassName
{
        ...
        …
        abstract Type MethodName1();
        …
        …
        Type Method2()
        {
           // method body
        }
}
```

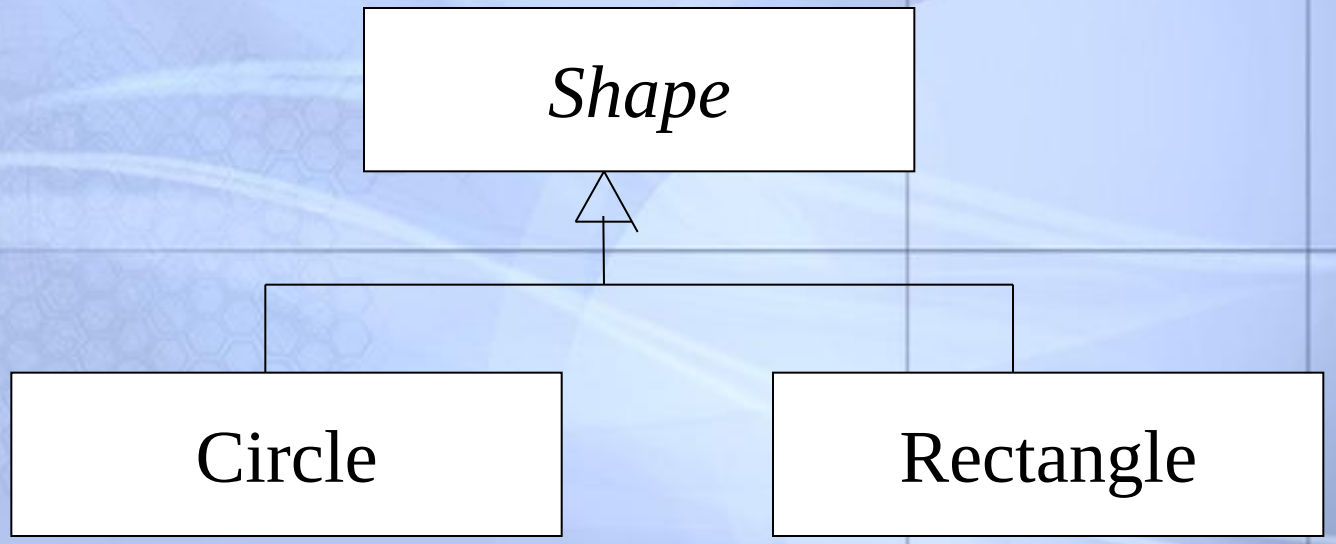
- When a class contains one or more abstract methods, it should be declared as abstract class.
- We cannot declare abstract constructors or abstract static methods.

Abstract classes and methods

- Abstract methods have **abstract** in the signature.
- Abstract methods have **no body**
- Abstract methods make the **class abstract**
- **Concrete subclasses complete the implementation**
- A concrete subclass **must** implement all abstract methods of its superclass
- An abstract class provides a high-level partial implementation of some concept
- Any subclass of an abstract class must either implement **all the abstract methods** in the superclass or be itself declared abstract.
- BUT – each concrete subclass can **implement the method differently**

Abstract Class Example

- Shape is a abstract class.



The Shape Abstract Class

```
public abstract class Shape {

        public abstract double area();
        public void move() { // non-abstract method
            // implementation
        }
}
```

- Is the following statement valid?
  - Shape s = new Shape();

- No. It is illegal because the Shape class is an abstract class, which cannot be instantiated to create its objects.

Abstract Class Example

- Classes Circle and Square are sub-classes of Shape;
- They must implement method area() declared as abstract in class Shape

```
public Circle extends Shape {

    protected double r;
    protected static final double PI =3.1415926535;
    public Circle() { r = 1.0; )
    public double area() { return PI * r * r; }
}
```

```
public Rectangle extends Shape {

    protected double w, h;
    public Rectangle() { w = 0.0; h=0.0; }
    public double area() { return w * h; }
}
```

Abstract class Example (cont'd)

```
public abstract  Animal{

    private String  nameOfAnimal;
    public abstract  void speak();
    public String  getAnimalName(){
        return nameOfAnimal;
    }

    public void setAnimalName(String name){
        nameOfAnimal = name;
    }
}
```

Abstract class Example (cont'd)

```
public class Dog extends Animal{

    public void speak() {
        System.out.println("Woof!");
    }
}
```

```
public class Cow extends Animal{

    public void speak() {

    System.out.println("Moo!");
    }
}
```

```
public class Snake extends Animal{

    public void speak() {

    System.out.println("Ssss!");
    }
}
```

Abstract class Example (cont'd)

```
public class  UseAnimals{

    public static void main (String[] args){
        Dog  myDog  = new Dog();
        Cow  myCow  = new Cow();
        Snake   mySnake  = new Snake();

        myDog.setAnimalName ("My dog Murphy");
        myCow.setAnimalName ("My cow Elsie");
        mySnake.setAnimalName ("My snake Sammy");

    System.out.print(myDog.getAnimalName() + "says" );
    myDog.speak();
    System.out.print(myCow.getAnimalName() + "says");
    myCow.speak();
    System.out.print(mySanke.getAnimalName() + "says" );
    mySnake.speak();
    }
}
```

Wrapper Class

Wrapper Classes

- Wrapper classes are used to manage primitive data as objects
- Each wrapper class represents a particular primitive type
- Wrappers: classes whose objects contain single values of the "wrapped type"
- Wrappers also contain other useful conversion operations (to / from String, etc.)

Wrapper Classes (cont.)

- The wrapper constuctors create class objects from the primitive types.

  Integer n = new Integer(42);

  Double aD = new Double(5.0);

  – Here a Double wrapper object is created by passing the double value in the Double constructor argument.

- To retrieve the integer and double value

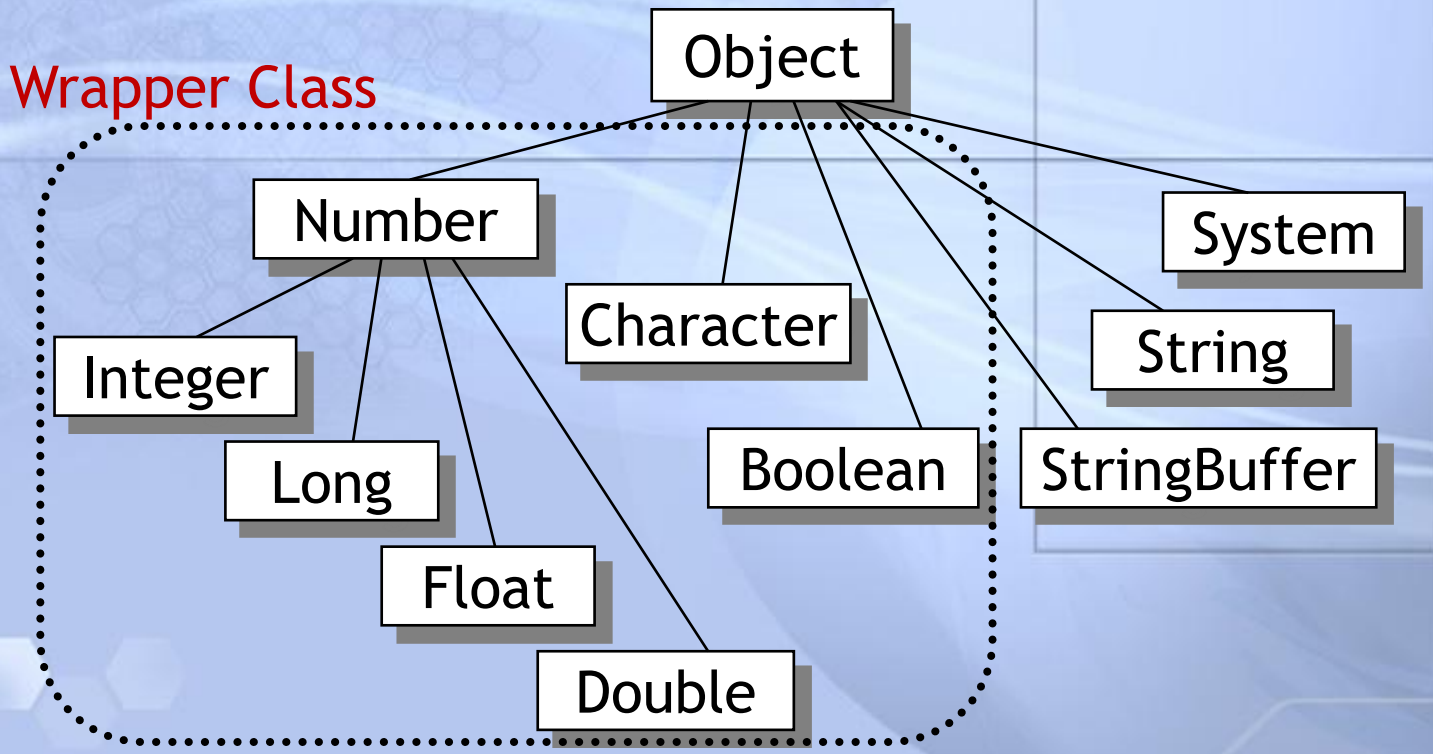
  int i = **n.intValue()**;

  double r = **aD.doubleValue();**

- Each wrapper has a similar method to access the primitive value: intValue() for Integer, booleanValue() for Boolean, and so forth.

Wrapper Class in java.lang Class

- All wrapper classes are part of the java.lang package: **Byte, Short, Integer, Long, Float, Double,Character, Boolean, Void**.



Wrapper Classes (cont.)

primitive type	wrapper class	extraction method
int	Integer	intValue
long	Long	longValue
float	Float	floatValue
double	Double	doubleValue
char	Character	charValue

Number Class

- Abstract Class
- Super Class of Integer, Long, Float, Double

- Method
  - **abstract int intValue()** : Convert into int type
  - **abstract long longValue()** : Convert into long type
  - **abstract float floatValue()** : Convert into float type
  - **abstract double doubleValue()** : Convert into double type

Integer Class

- Constant
  - public static final int MAX_VALUE = 2147483647
  - public static final int MIN_VALUE = -2147483648
- Method
  - **static int parseInt(String s)** :
    - Convert a Number in String into int type
  - **static int parseInt(String s , int radix)** :
    - Convert a number in String into int type with radix
  - **static String toBinaryString(int i)** :
    - Convert into binary string form
  - **static String toHexString(int i)** :
    - Convert into hexadecimal string form

```
Interger.parseInt(s);
Interger.toBinaryString(i);
. . .
```

Double Class

- Constant
  - –public static final double MAX_VALUE=1.79769313486231570e+308
  - – public static final double MIN_VALUE= 4.94065645841246544e-308
  - – public static final double NaN = 0.0 / 0.0
  - – public static final double NEGATIVE_INFINITY = -1.0 / 0.0
  - – public static final double POSITIVE_INFINITY = 1.0 / 0.0
- **static boolean isInfinite(double v)** :
  - – Check whether the parameter is infinite or not.
  - – **static Double valueOf(String s)** : Method
  - – **static long doubleToLongBits(double value)** :
    - Convert the bits represented by double type into long type bit pattern

Boolean Class

- Constant
  - public static final Boolean TRUE = new Boolean(true)
  - public static final Boolean FALSE = new Boolean(false)

- Method
  - **Boolean(boolean b)** :
    - Constructor to create boolean object receiving the initial value b
  - **Boolean(String s)** :
    - Constructor to receive the string value "true“ or "false“
  - **boolean booleanValue()** :
    - Return the boolean value of object
  - **static boolean getBoolean(String name)** :
    - Return the boolean value of system attribute
  - **static Boolean valueOf(String s)** :
    - Return the Boolean value correspond to string s

Character Class

- Constant
  - public static final int MAX_RADIX = 36
  - public static final char MAX_VALUE =‘\ffff’
  - public static final int MIN_RADIX = 2
  - public static final char MIN_VALUE =’\0000’

- Method
  - **Character(char value)** : Constructor to initialize the object as value
  - **char charValue()** : Convert into char type
  - **static boolean isDigit(char ch)** : Test whether is digit?
  - **static boolean isLetter(char ch)** : Test whether is letter?
  - **static boolean isLetterOrDigit(char ch)** : Return when it is letter or digit.

Thank You!