Chapter 8

Using Predefined Classes

Java Class Library

Large collection of predefined classes you can use in your programs.

Also called the Application Program Interface (API)

`import` Statement

```
import java.util.Random;

import java.util.*;
```

String class

```
String s;
s = new String("hello").
```

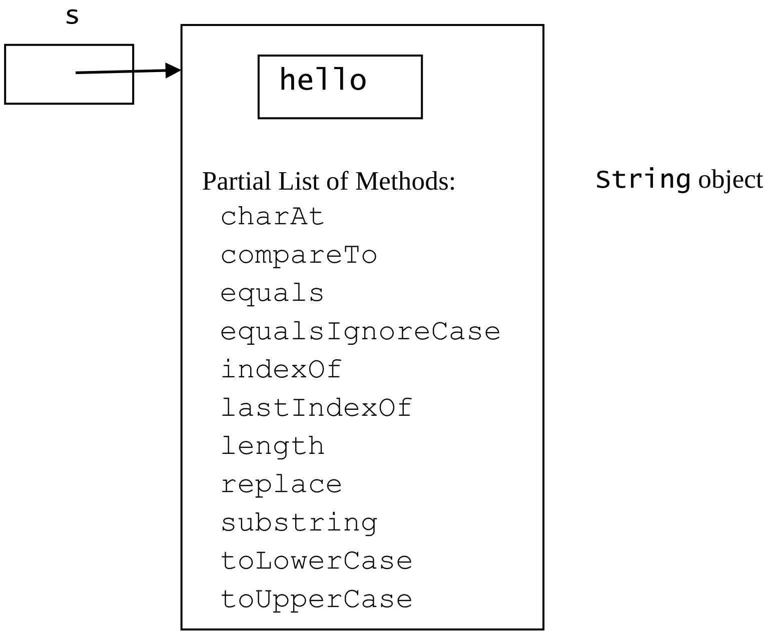
or

```
String s = new String("hello");
```

or

```
String s = "hello";
```

A typical `String` object



s

hello

Partial List of Methods:

- `charAt`
- `compareTo`
- `equals`
- `equalsIgnoreCase`
- `indexOf`
- `lastIndexOf`
- `length`
- `replace`
- `substring`
- `toLowerCase`
- `toUpperCase`

`String` object

charAt

```
String s = "hello";

// returns char at index 1
char c = s.charAt(1);
```

compareTo

```
if (s1.compareTo(s2) < 0)
{
   System.out.println(s1);
   System.out.println(s2);
}
else
{
   System.out.println(s2);
   System.out.println(s1);
}
```

`equals`

```
if (s1.equals(s2))
   System.out.println("They are equal");
else
   System.out.println("They are not equal");
```

Wrong:

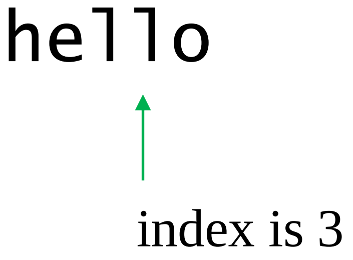
```
if (s1 == s2)
   System.out.println("They are equal");
else
   System.out.println("They are not equal");
```

`equalsIgnoreCase` works like `equals`

indexOf

```
int i = s.indexOf("lo");
```

`lastIndexOf` works like `indexOf`.

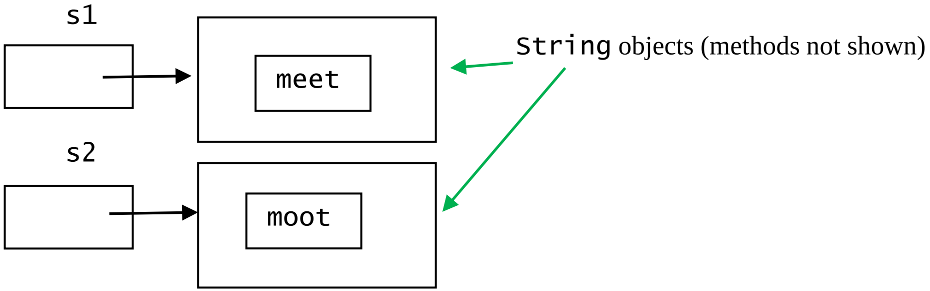


hello

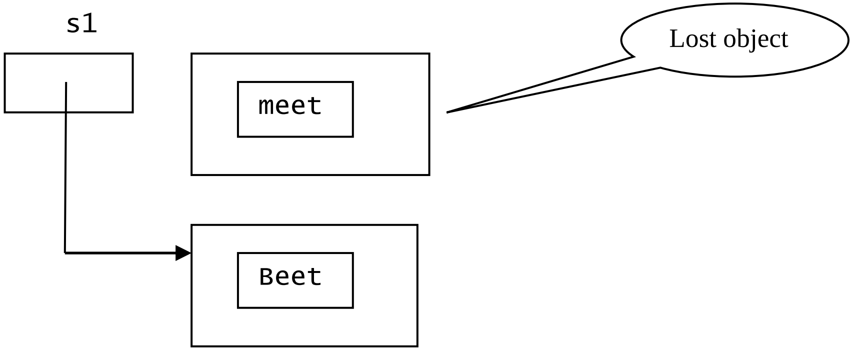
index is 3

replace

```
s2 = s1.replace('e', 'o');
```



```
s1 = s1.replace('m', 'B');
```



substring

```
s2 = s1.substring(1);

s2 = s1.substring(1, 4);
```

length

```
i = s.length();
```

toLowercase

```
s2 = s1.toLowerCase();
```

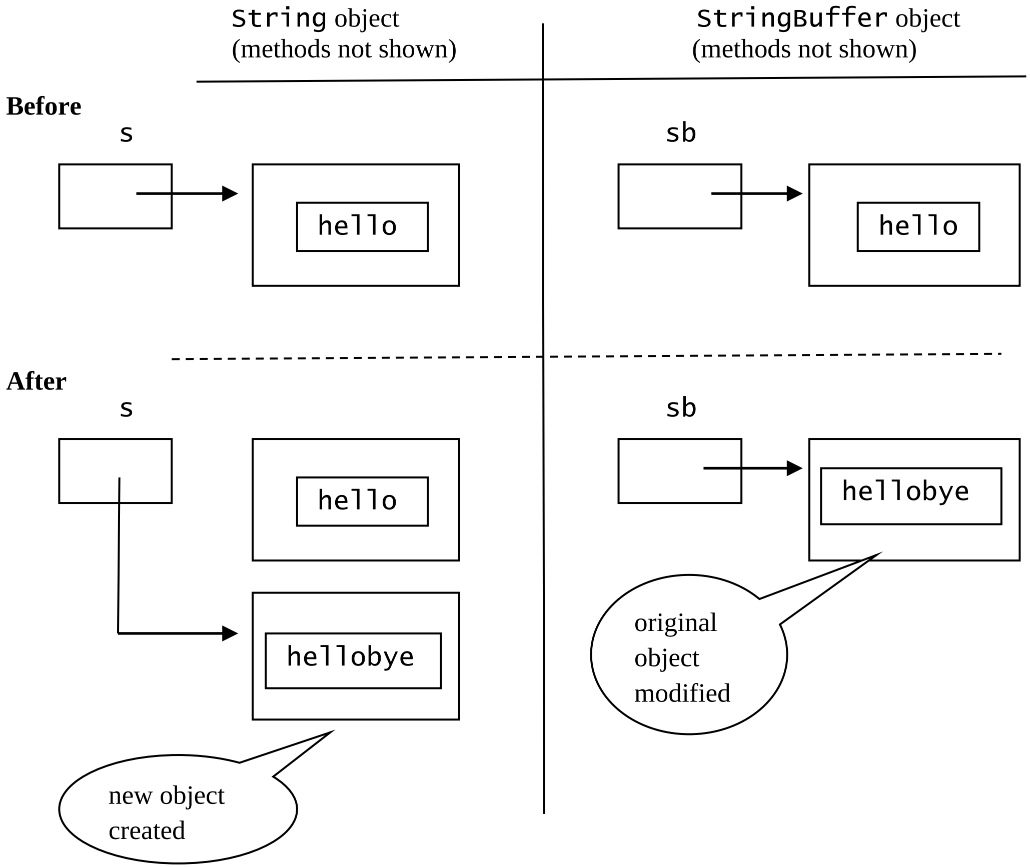
StringBuffer class

```
class StringBufferExample
{
   public static void main(String[] args)
   {
      String s;
      StringBuffer sb;
      s = "hello";
      sb = "hello";       // illegal
      sb = new StringBuffer("hello");
      s = sb.toString();
```

```
        // would work if sb were a String
        System.out.println(sb);
        System.out.println(sb.charAt(0));
        System.out.println(sb.length());
        System.out.println(sb.substring(1));
        System.out.println(sb.substring(1, 3));
        System.out.println(sb.indexOf("lo"));
```

Stringbuffer append does not create a new object

```
// append differently
sb.append("bye");       // hellobye
s = s + "bye";          // hellobye
```


String object
(methods not shown)
StringBuffer object
(methods not shown)
Before
s
hello
sb
hello
After
s
hello
hellobye
new object
created
sb
hellobye
original
object
modified

```
        // would not work if sb were a String
        sb.insert(5, " / ");   // hello / bye
        sb.setCharAt(0, 'm');  // mello / bye
        sb.deleteCharAt(6);    // mello  bye
        sb.delete(2, 4);       // meo  bye
        sb.replace(3, 8, "w"); // meow
        sb.reverse();          // woem
```

Math Class

`public static double abs(int x)`	absolute value
`public static double abs(long x)`	absolute value
`public static double abs(float x)`	absolute value
`public static double abs(double x)`	absolute value
`public static double ceil(double x)`	smallest whole number $\geq x$
`public static double floor(double x)`	largest whole number $\leq x$
`public static long round(double x)`	x rounded up to long
`public static double sin(double x)`	sin(x), x in radians
`public static double cos(double x)`	cos(x), x in radians
`public static double tan(double x)`	tan(x), x in radians
`public static double exp(double x)`	e^x
`public static double pow(double x, double y)`	x^y
`public static double sqrt(double x)`	square root of x
`public static double max(int x, int y)`	larger of x and y
`public static double max(double x, double y)`	larger of x and y
`public static double min(int x, int y)`	smaller of x and y
`public static double min(double x, double y)`	smaller of x and y

`Random` Class

Constructors

```
public Random()
```

Constructs a `Random` object using the time of day as the seed.

```
public Random(long s)
```

Constructs a `Random` object using the value of the parameter `s` as the seed.

Some methods in the `Random` class

`public boolean nextBoolean()`

Returns `true` or `false`, uniformly distributed.

`public double nextDouble()`

Returns a `double` pseudo-random number between 0.0 (inclusive) and 1.0 (exclusive). The numbers are uniformly distributed.

`public float nextFloat()`

Returns a `float` pseudo-random number between 0.0 (inclusive) and 1.0 (exclusive). The numbers are uniformly distributed.

`public int nextInt()`

Returns an `int` pseudo-random number. The numbers are uniformly distributed.

`public int nextInt(int n)`

Returns an `int` pseudo-random number of type `int` between 0 and n - 1, inclusive. The numbers are uniformly distributed over the interval 0 to n - 1.

`public long nextLong()`

Returns a `long` pseudo-random number of type `long`. The numbers are uniformly distributed.

`public void setSeed(long s)`

Sets the seed to `s`.

Illustrative program that uses Random

```
import java.util.Random;
class TestRandom
{
   public static void main(String[] args)
   {
        Random r1 = new Random(7777777);
        System.out.println("r1 object");
        System.out.println(r1.nextDouble());
        System.out.println(r1.nextDouble());

        System.out.println(r1.nextInt());
        System.out.println(r1.nextInt());

        System.out.println(r1.nextInt(2));
        System.out.println(r1.nextInt(2));
```

Another object with same seed

```
        Random r2 = new Random(7777777);
        System.out.println("r2 object");
        System.out.println(r2.nextDouble());
        System.out.println(r2.nextDouble());

        System.out.println(r2.nextInt());
        System.out.println(r2.nextInt());

        System.out.println(r2.nextInt(2));
        System.out.println(r2.nextInt(2));
   }
}
```

Output

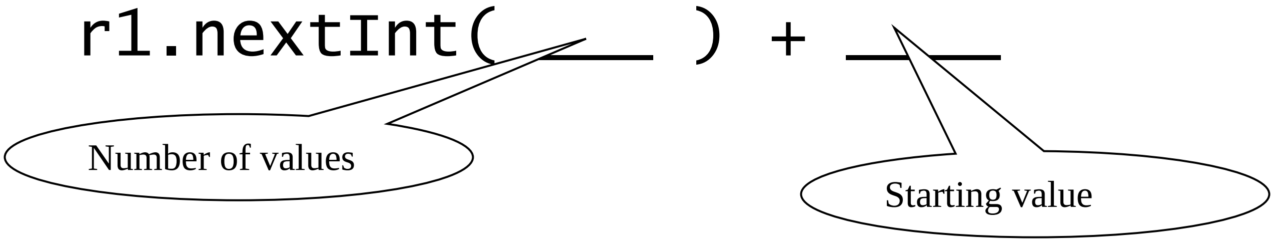
```
r1 object
0.7748014570608913
0.2410579504844711
875655393
495956042
0
1
r2 object
0.7748014570608913
0.2410579504844711
875655393
495956042
0
1
```

Modelling coins, dice, grades

```
coinToss = r1.nextInt(2);

dieThrow = r1.nextInt(6) + 1;

grade = r1.nextInt(41) + 60;
```



```
r1.nextInt( ____ ) + ____
```

Number of values

Starting value

Bell-shaped Curve

```
grade =  (r1.nextInt(41) + 60 +
          r1.nextInt(41) + 60)/2;
```

or

```
grade = (r1.nextInt(41) +
         r1.nextInt(41))/2 + 60;
```

Scanner Class

```java
import java.util.Scanner;
class Average
{
   public static void main(String[] args)
   {
      Scanner kb = new Scanner(System.in);
      int sum = 0.0;
      int numberOfGrades, i = 1;

      System.out.println("Enter number of grades");
      numberOfGrades = kb.nextInt();
      System.out.println("Enter grades");

      while (i <= numberOfGrades)
      {
         sum = sum + kb.nextInt();
         i++;
      }

      System.out.println("Avg = "+ (double) sum/numberOfGrades);
   }
}
```

Use loop to count grades

```
double sum = 0.0;
int numberOfGrades = 0;
while (true)
{
   double x;
   x = kb.nextDouble();
   if (x < 0.0) break;
   sum = sum + x;
   numberOfGrades++;
}
```

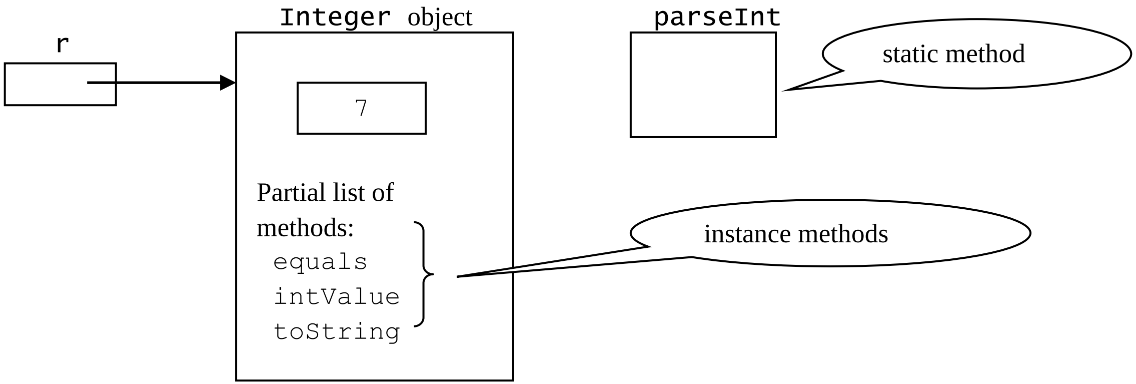
Wrapper Classes

Primitive Type	Corresponding Wrapper Class
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`

```
Integer r;
r = new Integer(7);
```

or

```
Integer r = new Integer(7);
```



Auto boxing and unboxing

```
Integer r = new Integer(5);
```

equivalent to

```
Integer r = 5;  // auto-box
```

```
int i = r.intValue();
```

equivalent to

```
int i = r;      // auto-unbox
```

Get data from wrapper class as `String`

```
Integer r;
String s;
r = 123;
s = r.toString();
```

Test for equality with equals

```
if (r1.equals(r2))
   System.out.println("Objects are equal");
```

Wrong:

```
if (r1 == r2))
   System.out.println("Objects are equal");
```

parseInt

parseInt converts String to int

```
String s = 717;
int i = Integer.parseInt(s);
```

Common features of wrapper classes

- They support auto-boxing and auto-unboxing.

- They have a `toString`, an `equals`, and a `compareTo` method. The `compareTo` method in the wrapper classes work like the `compareTo` method in the `String` class. Specifically, it returns an integer that is either less than zero, zero, or greater that zero to indicate the result of the compare.

- They are immutable. Once an object is created from a wrapper class, it cannot be changed.

- They have a method comparable to `intValue` in `Integer` but named differently. For example, `Double` has `doubleValue`. These methods return the primitive data in the wrapper object. However, we normally do not have to use them because auto-unboxing extracts the primitive data in a wrapper object for us.

Static methods in `char`

```
public static boolean isDigit(char c)
public static boolean isLetter(char c)
public static boolean isLetterOrDigit(char c)
public static boolean isLowerCase(char c)
public static boolean isUpperCase(char c)
public static boolean isWhitespace(char c)
public static char    toLowerCase(char c)
public static char    toUpperCase(char c)
```

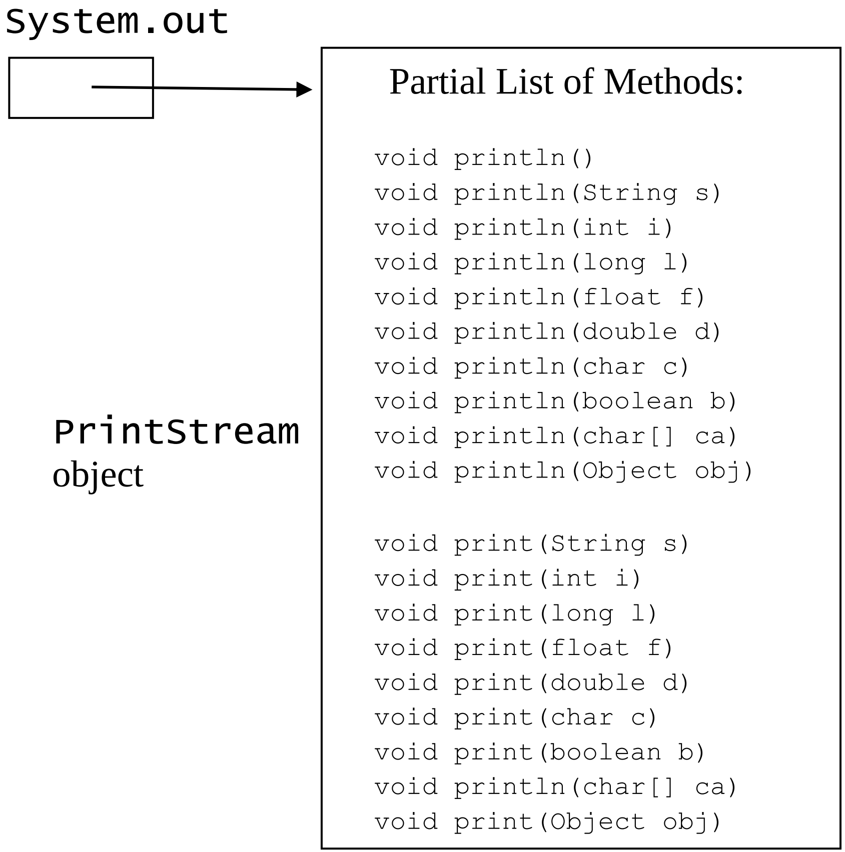
Range constants in wrapper classes

`MIN_VAL`
`MAX_VAL`

In `Byte`:

`Byte.MIN_VAL` is -128 (the smallest value of type `byte`)
`Byte.MAX_VAL` is 127 (the largest value of type `byte`).

System and System.out



System.out

PrintStream
object

Partial List of Methods:

```
void println()
void println(String s)
void println(int i)
void println(long l)
void println(float f)
void println(double d)
void println(char c)
void println(boolean b)
void println(char[] ca)
void println(Object obj)

void print(String s)
void print(int i)
void print(long l)
void print(float f)
void print(double d)
void print(char c)
void print(boolean b)
void println(char[] ca)
void print(Object obj)
```