Session 02
Learning the Java Language

(http://docs.oracle.com/javase/tutorial/java/index.html)

Trinh Thi Van Anh – PTIT

Objectives

- Study some fundamentals of Java languages: Data types, variables, arrays, operators,…

- The Basic Constructs: loop, if, swidch()….

- Wrapper Class

- Input/output variables

- String class

- Regular Expression

- Stringbuilder, stringbuffer, StringTokenizer

Keywords and Identifiers

- Keywords: Almost of them are similar to those in C language

- Variables in Java are case sensitive and can be of unlimited sequence of letters and numbers. However, variable names must start with a letter, underscore character "_", or a dollar sign "$".

- Identifiers must be different to keywords

An_Identifier
a_2nd_Identifier
Go2
$10

An-Identifier
2nd_Identifier
goto
10$

Naming conventions in java

- class name: should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.

- interface name: should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.

- method name: should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.

- variable name: should start with lowercase letter e.g. firstName, orderNumber etc.

- package name: should be in lowercase letter e.g. java, lang, sql, util etc.

- constants name: should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

Primitive Data Types - Variables

- A *primitive* is <u>a simple non-object</u> data type that represents a single value. Java's primitive data types are:

Type	Bytes	Minimum	Maximum
char	2	\u0000	\uFFFF
byte	1	-2^7	$2^7 - 1$
short	2	-2^{15}	$2^{15} - 1$
int	4	-2^{31}	$2^{31} - 1$
long	8	-2^{63}	$2^{63} - 1$
float	4		
double	8		
boolean	true/false		

Type   var [=Initial value] ;

Operators

Category (Descending Precedence)	Operators
Unary	`++ -- + - ! ~ (type)`
Arithmetic	`* / %` `+ -`
Shift	`<< >> >>>`
Comparison	`< <= > >= instanceof` `== !=`
Bitwise	`& ^ |`
Short-circuit	`&& ||`
Conditional	`?:`
Assignment	`= op=`

They are the same with those in C language

Using Operators Demonstration


```java
public class UseOps {
    public static void main(String[] args)
    {   int x=-1;
        System.out.println("-1<<1: " + (x<<1) );
        System.out.println("-1>>1: " + (x>>1) );
        System.out.println("-1>>>1: " + (x>>>1));
        System.out.println("3|4: " + (3|4));
        System.out.println("3&4: " + (3&4));
        System.out.println("3^4: " + (3^4));
        String S="Hello";
        boolean result= S instanceof String;
        System.out.println("Hello is an instance of String: " + result);
    }
}
```

```
Output - Chapter01 (run)
run:
-1<<1: -2
-1>>1: -1
-1>>>1: 2147483647
3|4: 7
3&4: 0
3^4: 7
Hello is an instance of String: true
BUILD SUCCESSFUL (total time: 0 seconds)
```

Literals and Value Variables

- Character: 'a'

- String: String S="Hello";

- Integral literals:

   28, 0x1c, 0X1A (default: int).
   123I, 123L (long)

- Floating point:

   1.234 (default: double)

   1.3f   1.3F

   1.3E+21

   1.3d   1.3D

Value variable

Stack

n   10

int n=10;

Escape sequence

\t   Insert a tab in the text at this point.

\b   Insert a backspace in the text at this point.

\n   Insert a newline in the text at this point.

\r   Insert a carriage return in the text at this point.

\f   Insert a formfeed in the text at this point.

\'   Insert a single quote character in the text at this point.

\"   Insert a double quote character in the text at this point.

\\   Insert a backslash character in the text at this point.

Java Expressions

- Java is an expression-oriented language. A simple expression in Java is either:
  - A constant: 7, false
  - A char - literal enclosed in single quotes: 'A', '3'
  - A String - literal enclosed in double quotes: "foo"
  - The name of any properly declared variables: x
  - Any two|one of the preceding types of expression that are combined with one of the Java binary operators: i++, x + 2, (x + 2)

Type Casting

- Assigning a value of one type to a variable of another type is known as **Type Casting**. In Java, type casting is classified into two types,

- Widening Casting(Implicit)

byte →short →int →long → float → double

widening
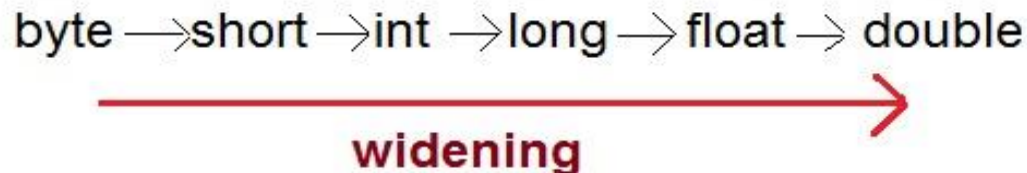
- Narrowing Casting(Explicitly done)

double → float → long → int → short → byte

Narrowing

Example casting

- double d = 100.04;

- long l = (long)d; //explicit type casting required int i = (int)l; //explicit type casting required

int a, b;
short c;
a = b + (int)c;
	int d;
	short e;
	e = (short)d;
		double f;
		long g;
		f = g;
		g = f; //error

One Dimensional Arrays (1)

- An *array* is a container object that holds a fixed number of values of a single type.

- The length of an array is established when the array is created.

- Each item in an array is called an *element*, and each element is accessed by its numerical *index.*
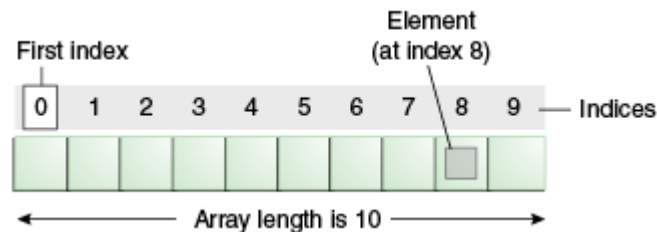
One Dimensional Arrays (2)

- Declaring a Variable to Refer to an Array

  ```
  int[] anArray;
  ```

  or `float anArrayOfFloats[];`

- Creating, Initializing, and Accessing an Array

  ```
  anArray = new int[10];
  ```

- Copying Arrays

  - Use arraycopy method from System class.

One Dimensional Arrays (3)

int[] ar;

ar= new int[3];
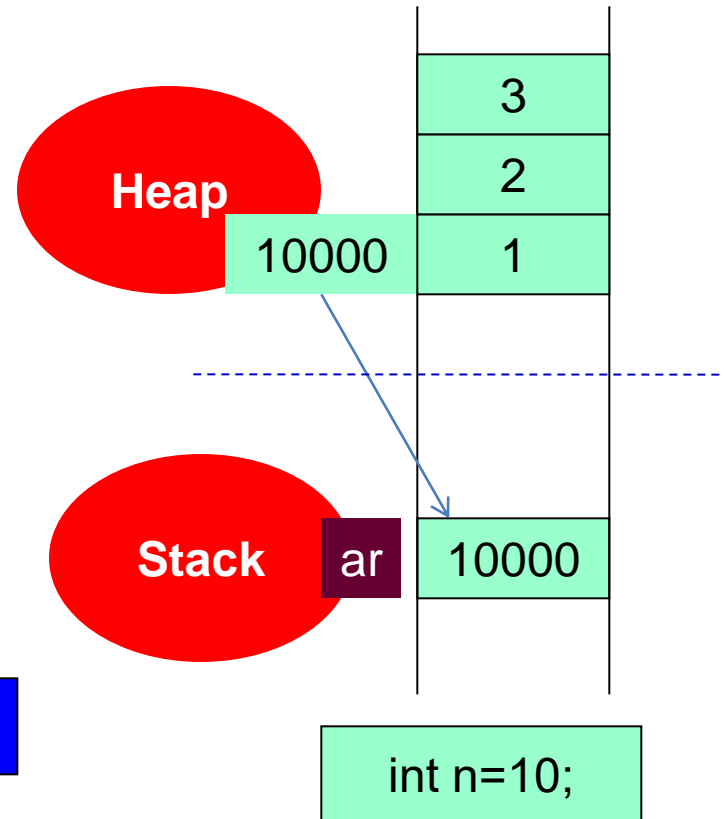
ar[0]=1; ar[1]=2; ar[2]=3;

int a2[];

int[] a3 = {1,2,3,4,5};

int a4[] = {1,2,3,4,5};

Array is a reference variable

Heap

3

2

10000 | 1

Stack | ar | 10000

int n=10;

example one dimensional array

```java
public class DaySo {
    int calSum(int... a){//int[] a
        int t=0;
        for(int x:a)
            t+=x;
        return t;
    }
    int calMin(int... a){
        int t=a[0];
        for(int x:a)
            if(t>x)
                t=x;
        return t;
    }
```

```java
int calMax(int... a){
        int t=a[0];
        for(int x:a)
            if(t<x)
                t=x;
        return t;
    }
int[] sort(int... a){
   int t;
   for(int i=0;i<a.length-1;i++)
     for(int j=i+1;j<a.length;j++)
       if(a[i]>a[j]){
         t=a[i];
         a[i]=a[j];
         a[j]=t;        }
   return a;
}
```
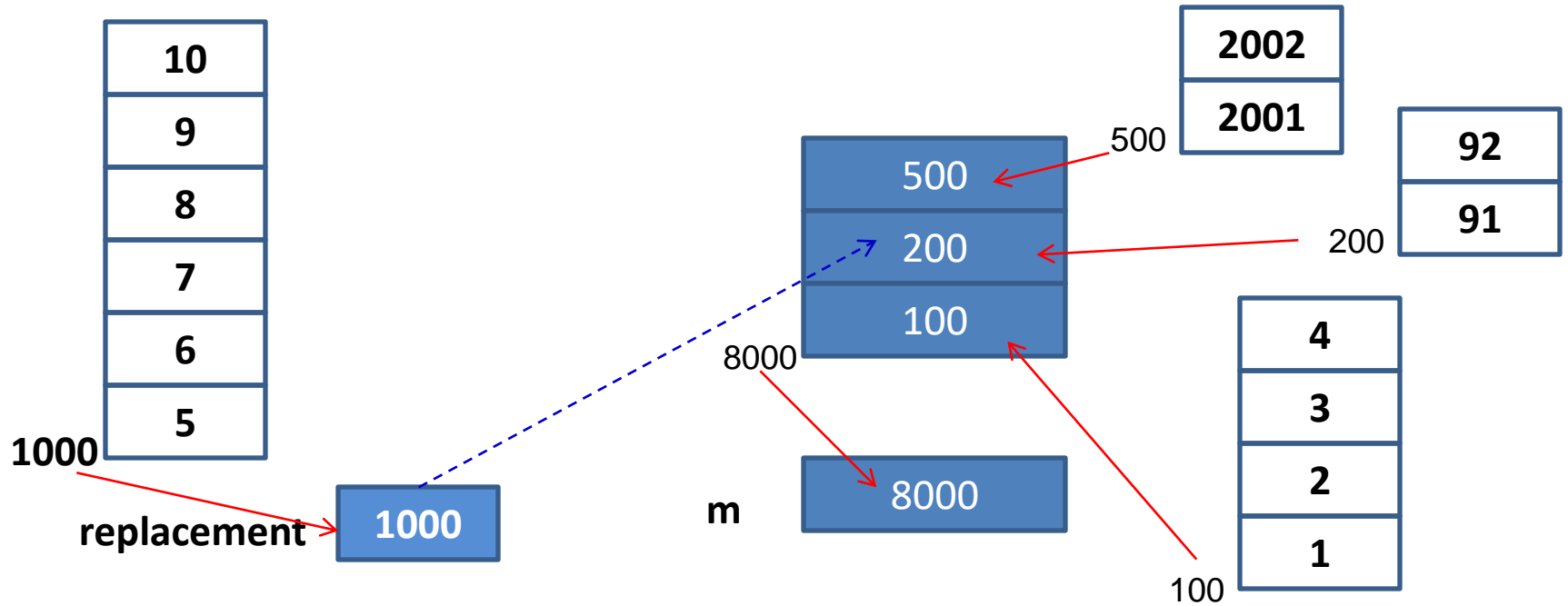
```java
int[] input(int n){
  Scanner in = new  canner(System.in);
  int[] b = new int[n];
  for(int i=0;i<n;i++){
    System.out.print("\n thu "+i+": ");
    b[i]=in.nextInt();
  }
  return b;      }
void toString(int... a){
  System.out.print("\n Day so: "+Arrays.toString(a));
}
```

Multiple Dimensional Arrays

int m[][]= { {1,2,3,4}, {91,92}, {2001,2002}};
int[] replacement = {5,6,7,8,9,10};
m[1]= replacement;

m[i][j]

```
int[][] m; // declare a matrix
int r=10, c=5; // number of rows, columns
m= new int[r][c]; // memory allocate
```

- `int b[3][4];`
- `int row = b.length;`
- `int col = b[0].length;`

to *add* elements to an *array*

```java
public double[][] add(double[][] a,
  double[][] b) {
    int m = a.length;
    int n = a[0].length;
    double[][] c = new double[m][n];
    for (int i = 0; i < m; i++)
      for (int j = 0; j < n; j++)
        c[i][j] = a[i][j] + b[i][j];
    return c;
}
```

to *Subtract*

```java
public double[][] subtract(double[][] a,
  double[][] b) {
  int m = a.length;
  int n = a[0].length;
  double[][] c = new double[m][n];
  for (int i = 0; i < m; i++)
    for (int j = 0; j < n; j++)
      c[i][j] = a[i][j] - b[i][j];
  return c;
}
```

To multiply

```java
public double[][] multiply(double[][] a,
   double[][] b) {
   int m1 = a.length;
   int n1 = a[0].length;
   int m2 = b.length;
   int n2 = b[0].length;
   if (n1 != m2) throw new
   RuntimeException("Illegal matrix dimensions.");
   double[][] c = new double[m1][n2];
   for (int i = 0; i < m1; i++)
     for (int j = 0; j < n2; j++)
       for (int k = 0; k < n1; k++)
         c[i][j] += a[i][k] * b[k][j];
    return c;
}
```
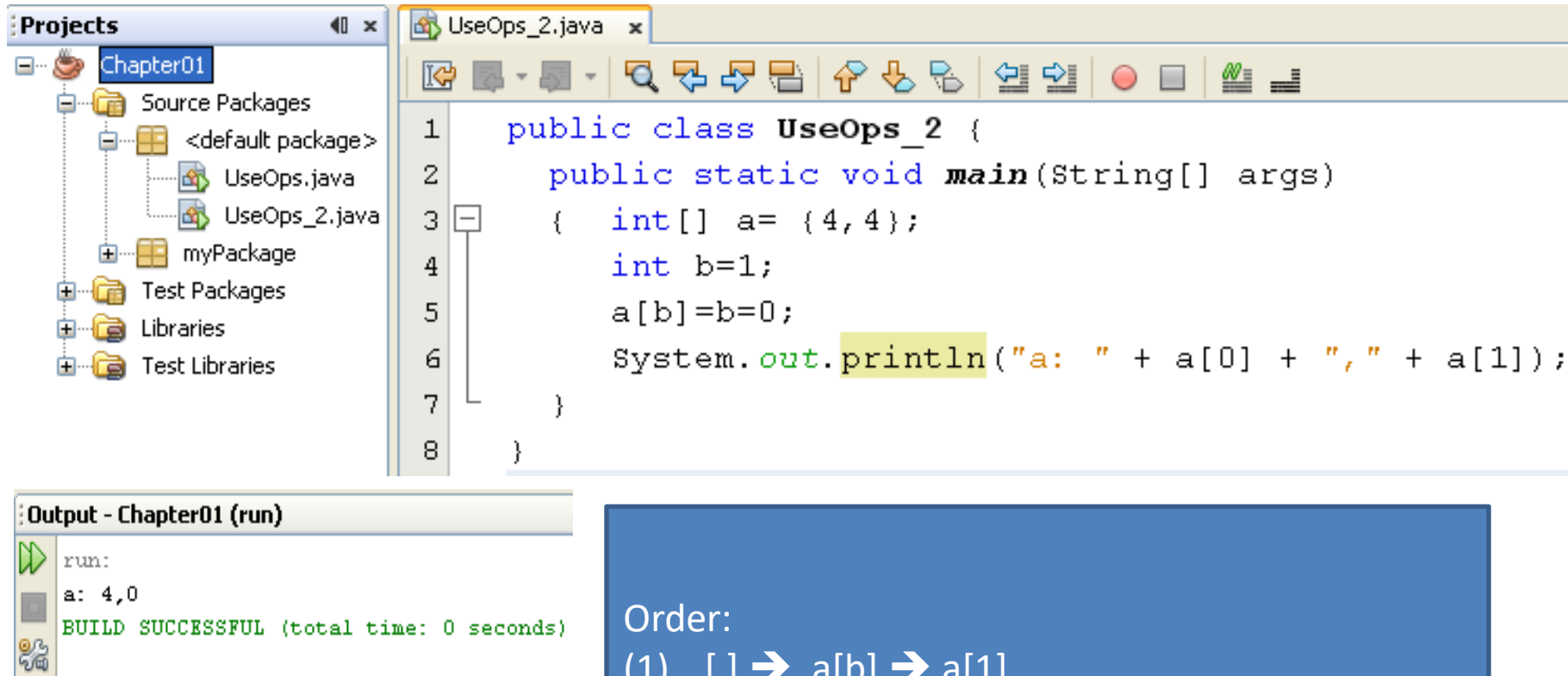
to *transpose* matrix

```java
public double[][]
  transpose(double[][] a) {
   int m = a.length;
   int n = a[0].length;
   double[][] b = new
  double[n][m];    for (int i = 0;
  i < m;  i++)
     for (int j = 0; j < n; j++)
       b[j][i] = a[i][j];
   return b;
}
```

Evaluating Expressions and Operator Precedence

- The compiler generally evaluates such expressions from the innermost to outermost parentheses, left to right.

```
int x = 1; int y = 2; int z = 3;
int answer = ((8 * (y + z)) + y) * x;
would be evaluated piece by piece as follows:
((8 * (y + z) ) + y) * x
((8 * 5) + y) * x
(40 + y) * x
42 * x
42
```

Operator Precedence- Evaluation Order

Projects

- Chapter01
  - Source Packages
    - \<default package\>
      - UseOps.java
      - UseOps_2.java
    - myPackage
  - Test Packages
  - Libraries
  - Test Libraries

UseOps_2.java

```java
public class UseOps_2 {
    public static void main(String[] args)
    {   int[] a= {4,4};
        int b=1;
        a[b]=b=0;
        System.out.println("a: " + a[0] + "," + a[1]);
    }
}
```

Output - Chapter01 (run)

```
run:
a: 4,0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Order:
(1)   [ ] ➔ a[b] ➔ a[1]
(2)   = (from the right) ➔ b=0 ➔ return 0
➔ a[1] = 0

The Basic Constructs

- Java provides three loop constructions. Taken from C and C++, these are:
  - while(),
  - do ,
  - for().

The *while()* Loop

```
while (boolean_condition)
{
    statement(s);
}
```

Example

```
int number = 1;
 while (number <= 200) {
     System.out.print(number + "
");
     number *= 2;
 }
```
Output:
```
 1 2 4 8 16 32 64 128
```

The do Loop

```
do {
    do_something
    do_more
} while (boolean_condition);
```

Example

```
// roll until we get a number other than 3
    Random rand = new Random();
    int die;
    do {
        die = rand.nextInt();
    } while (die == 3);
```

The *for()* Loop

```
for (start_expr;
  test_expr;increment_expr){
    // code to execute repeatedly
    }
```

Example

- ```
  for (int index = 0; index < 10;
  index++) {
   System.out.println(index);
   }
  ```
- ```
  for (int i = -3; i <= 2; i++) {
      System.out.println(i);
     }
  ```
- **```
  for (int i = 3; i >= 1; i--) {
  ```**
  ```
      System.out.println(i);
   }
  ```

# Enhanced *for* Loops

- Java's for loops were enhanced in release 1.5 to work more easily with arrays and collections.

- Syntax:

  for (type variable_name:array)

```
int sumOfLengths(String[] strings) {
    int totalLength = 0;
    for (String s:strings)
            totalLength += s.length();
    return totalLength;
}
```

Example

```java
public class Example{
  public static void main(String args[]){
      int [] numbers = {10, 20, 30, 40, 50};
      for(int x : numbers ){
          System.out.print( x );// numbers[i]
          System.out.print(",");
      }
      System.out.print("\n");
     String [] names ={"James", "Larry", "Tom",
   "Lacy"};
      for( String name : names ) {
          System.out.print( name );
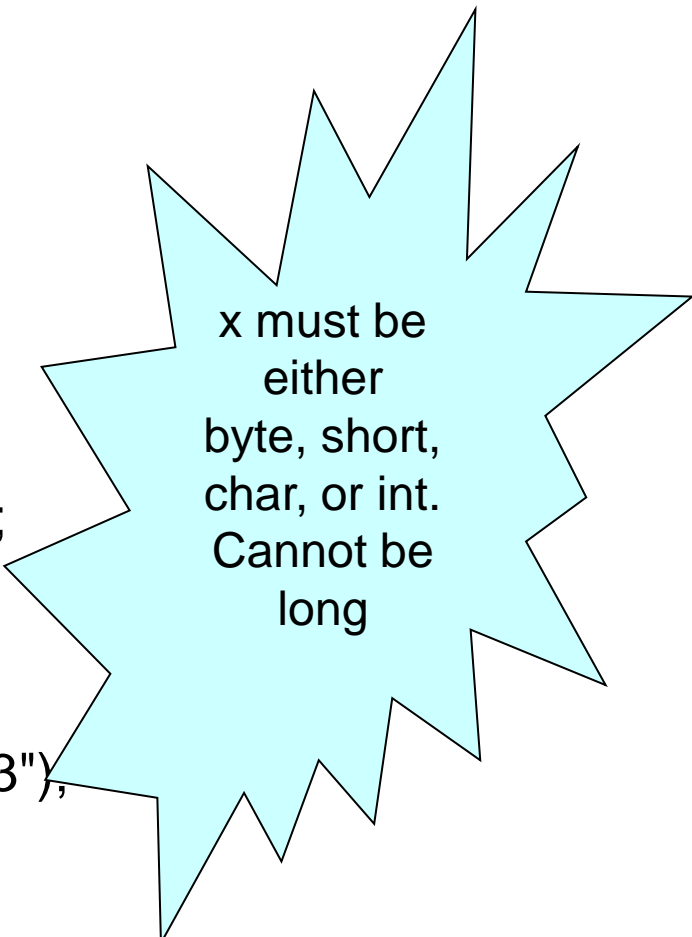          System.out.print(",");
      } } }
```

- Output:
  ```
  10,20,30,40,50,
  James,Larry,Tom,Lacy,
  ```

The Selection Statements

- ## The *if()/else* Construct

- ## The *switch()* Construct

```
switch (x) {
        case 1:
                System.out.println("Got a 1");
                break;
        case 2:
        case 3:
                System.out.println("Got 2 or 3");
                break;
        default:
                System.out.println("Not a 1, 2, or 3");
                break;
}
```

x must be either byte, short, char, or int. Cannot be long

If example

```java
class CheckNumber{
 public static void main(String
  args[]){
  int num =10;
  if (num %2 == 0)
    System.out.println(num+" is even");
  else
    System.out.println (num+" is odd");
  }
}
```

```java
public class SwitchDemo {
  public static void main(String[] args){
      int m = 8;
      String month;
      switch(m){
         case 1: month = " January "; break;
         case 2: month = " February "; break;
         case 3: month = " March"; break;
         case 4: month = " April"; break;
         case 5: month = " May"; break;
         case 6: month = " June"; break;
         case 7: month = " July"; break;
         case 8: month = " August"; break;
         case 9: month = " September"; break;
```

```java
        case 10: month = " October"; break
        case 11: month = " November"; break;
        case 12: month = " December"; break;
        default: month = "not a month";
                 break;
    }
    System.out.println(month);
    }
}
```

The *continue* Statements in Loops (for, while, do)

```
for( .;.;.)
{
    //process part 1
    if(condition)
        continue;
    //process part 2
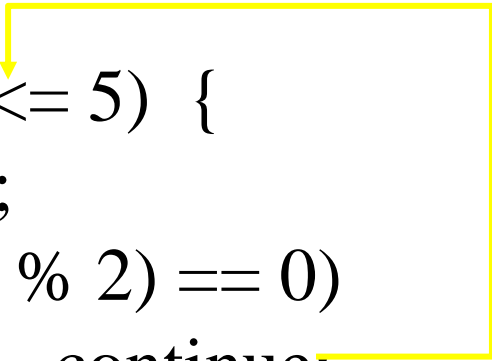}
```

```
mainloop:for(.;.;.){
    for( .;.;.)
    {
        //process part 1
        if(boolean_exp)
            continue mainloop;
        //process part 2
    }
}
```

continue

```
for (i=0; i<=5; ++i) {
    if (i % 2 == 0)
        continue;
    System.out.println("This is a " + i + " iteration");
}
```

```
i = 0;
  while (i <= 5)  {
        ++i;
        if (i % 2) == 0)
            continue;
        System.out.println("This is a odd iteration - " + i);
  }
}
```

The *break* Statements in Loops (for, while, do)

```
for( .;.;.)
{
    //process part 1
    if(condition)
        break;
    //process part 2
}
```

Break

```
int i = 1;
 while (true) {
      if (i = = 3)
            break;
      System.out.println("This is a " + i + " iteration");
      ++i;
 }
```

Menu

```java
while(true){
    System.out.print("\n 1.Select 1");
    System.out.print("\n 2.Select 2");
    System.out.print("\n 3.Select 3");
    System.out.print("\n 0.Exit");
    System.out.print("\n Choose 1,2,3
or 0: ");
    Scanner in = new
Scanner(System.in);
    int c = in.nextInt();
    System.out.print("\n");
```

```
switch(c){
    case 1: //to do for selecting 1
            break;
    case 2://to do for selecting 2
            break;
    case 3: //to do for selecting 3
            break;
    case 0: System.out.print("\n Bye!!!");
            System.exit(0);
            break;
    default:
 System.out.print("\n Choose 1,2,3 or 0 ");
    }
}
```

Basic Logic Constructs example

- They are the same with those in C-statements

An enhanced for loop

```
 2      package com;
 🔒⊟    import java.lang.*;
 4      public class Chao {
 5 ⊟      public static void main(String args[]){
 6              System.out.println("Hello");
 7              int a[] = { 1,2,3,4,5};
 8              for (int i=0;i<a.length;i++)System.out.print(a[i] + ",");
 9              System.out.println();
10              for (int x : a)  System.out.print(x + ",");
11              System.out.println();
12              for (int x : a) x+=10;
13              for (int i=0;i<a.length;i++)System.out.print(a[i] + ",");
14              System.out.println();
15        }
16      }
```

Read only

a | 1 | 2 | 3 | 4 | 5

x | 1
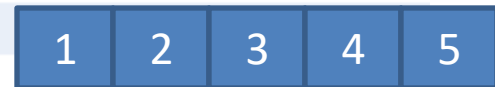
Output – P1 (run)

```
run:
Hello
1,2,3,4,5,
1,2,3,4,5,
1,2,3,4,5,
```

The String type

- A String represents a sequence of zero or more Unicode characters.
    - String name = "Steve";
    - String s = "";
    - String s = null;
- String concatenation.
    - String x = "foo" + "bar" + "!";
- Java is a case-sensitive language.

Type Conversions and Explicit Casting

```java
public class Casting_Convert_1 {
    public static void main (String[] args)
    {   short x, y = 256;
        byte m,n = 6;
        x = n ;  // Systematic Conversion
        n = y;   // narrow conversion
        n = (byte) y; // narrow casting, possible loss of precision
        System.out.println(n);
    }
}
```

```java
public class Casting_Convert_1 {
    public static void main (String[] args)
    {   short x, y = 256;
        byte m,n = 6;
        x = n ;  // Systematic Conversion
        n = (byte) y; // narrow casting, possible loss of p
        System.out.println(n);
    }
}
```

Output - Chapter04 (run)

```
run:
0
BUILD SUCCESSFUL (total time: 0 seconds)
```

* Widening Conversion: OK
- Narrowing conversion: Not allowed. We must use explicit casting.
- A boolean can not be converted to any other type.
- A non-boolean can be converted to another non-boolean type.

0000 0001

0000 0000

y

n

Scope of a Variable

ScopeDemo.java

```java
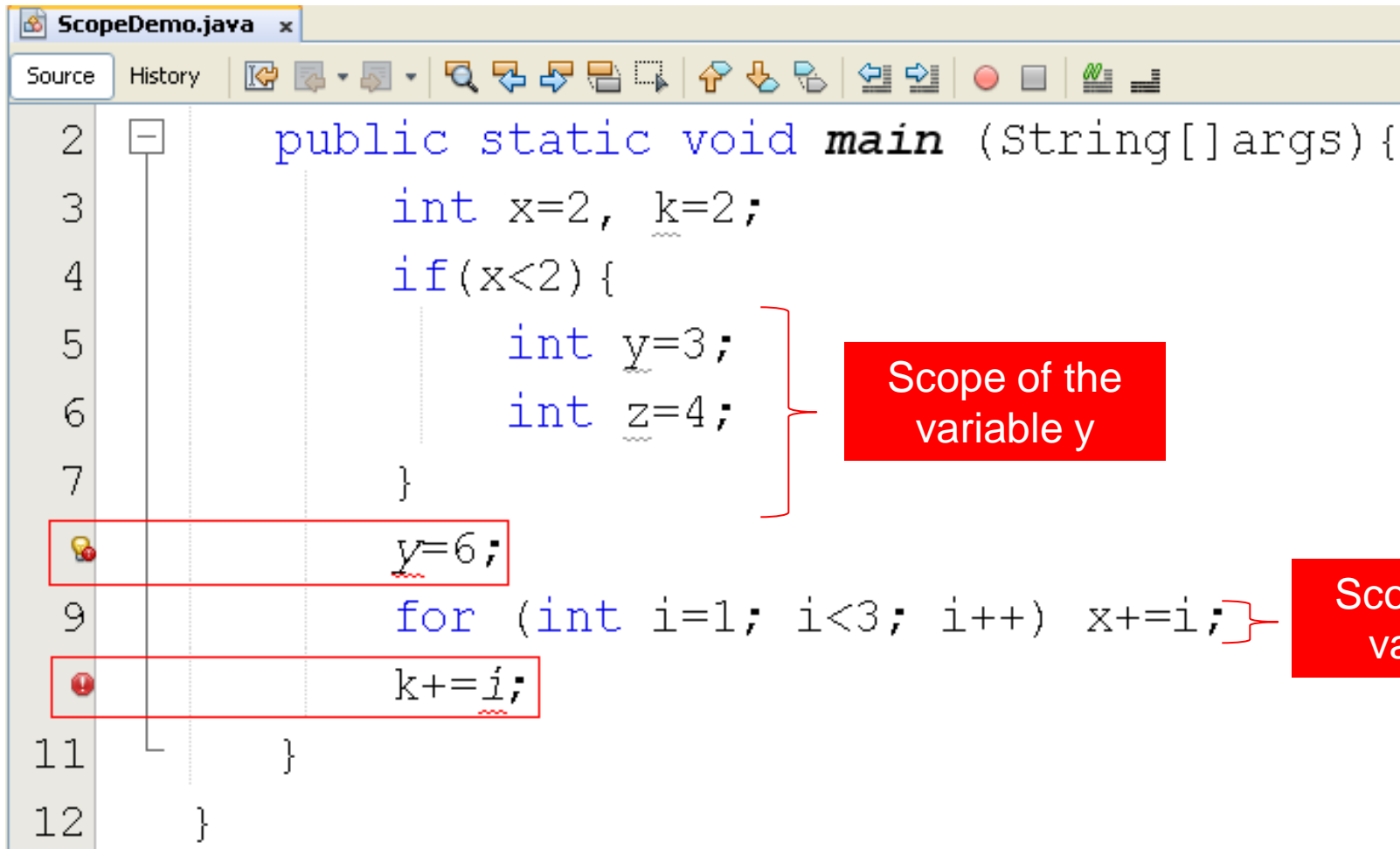public static void main (String[]args){
    int x=2, k=2;
    if(x<2){
        int y=3;
        int z=4;
    }
    y=6;
    for (int i=1; i<3; i++) x+=i;
    k+=i;
    }
}
```

Scope of the variable y

Scope of the variable i

Random class

- `import  java.util.Random;`
- `Random rd=new Random();`
- `int a=rd.nextInt(n);//0-(n-1)`
- `int t = rd.nextInt(b-a+1) +a;//(from a to b)`
- `Float =rd.nextFloat();// 0-1`

Math class

- java.lang
- Math.PI
- Math.abs(-20);
- double c = Math.ceil(7.342); // 8.0
- double f = Math.floor(7.343); // 7.0
- double p = Math.pow(2, 3); // 8.0
- double s = Math.sin(Math.PI/2);//1
- double a = Math.sqrt(9);//3
- ……

Wrapper Class

- `java.lang`

Primitive Type	Size	Minimum Value	Maximum Value	Wrapper Type
char	16-bit	Unicode 0	Unicode $2^{16}-1$	Character
byte	8-bit	-128	+127	Byte
short	16-bit	-2^{15} (-32,768)	$+2^{15}-1$ (32,767)	Short
int	32-bit	-2^{31} (-2,147,483,648)	$+2^{31}-1$ (2,147,483,647)	Integer
long	64-bit	-2^{63} (-9,223,372,036,854,775,808)	$+2^{63}-1$ (9,223,372,036,854,775,807)	Long
float	32-bit	32-bit IEEE 754 floating-point numbers		Float
double	64-bit	64-bit IEEE 754 floating-point numbers		Double
boolean	1-bit	`true` or `false`		Boolean
void	-----	-----	-----	Void

Autoboxing and Unboxing (1)

- java 5.0 introduces two very simple but convenient functions that <u>unwrap wrapper</u> objects and <u>wrap up</u> primitives.

- Converting a primitive value into an object of the corresponding wrapper class is called autoboxing.

- Converting an object of a wrapper type to its corresponding primitive value is called unboxing.

Primitive type		Wrapper class
boolean		Boolean
byte	**Boxing**	Byte
char	→	Character
float		Float
int	←	Integer
long	**Unboxing**	Long
short		Short
double		Double

Output???

- `Integer y = 567; // make a wrapper`
- `Integer y = new Integer(567);`
- `Integer x = y; //assign a second ref`

```
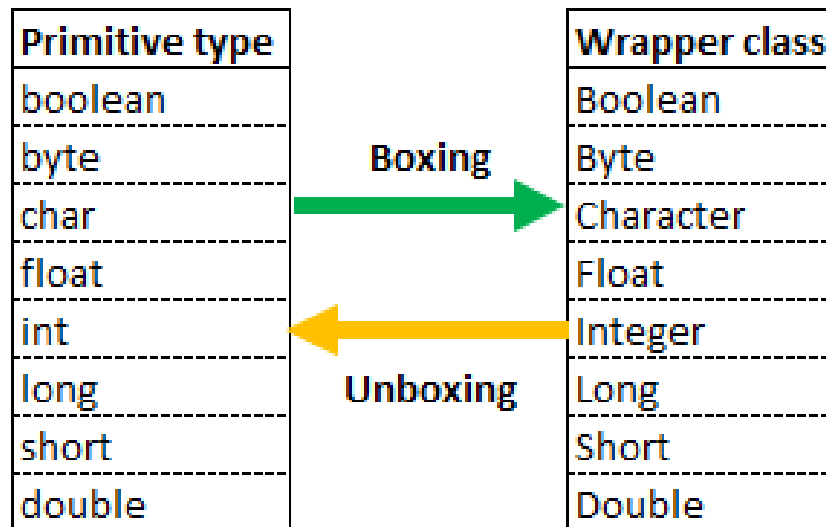        // var to THE wrapper
  System.out.println(y==x);

   y++; // unwrap, use, "rewrap"
  System.out.println(x + " " + y);

   System.out.println(y==x);
```

Autoboxing and Unboxing…(2)

- Sample of autoboxing and unboxing

```java
Integer wrappedInt = 25; //boxing or auto boxing

Double area(double radius) {
    return Math.PI * radius * radius; //boxing
}

Integer wi = 234;
int times9 = wi * 9; //unboxing
```

Return primitive types

- To return primitive types: using method **`typeValue()`**

```
// make a new wrapper object
Integer i2 = new Integer(42);
// byte
byte b = i2.byteValue();
// short
short s = i2.shortValue();
// double
double d = i2.doubleValue();
```

to convert from String to a primitive type

- **Using methods (static) of wrapper classes**

  ```
  static <type> parseType(String s)
  ```

- **Example**

  ```
  String s = "123";
  // int
  int i = Integer.parseInt(s);
  // short
  short j = Short.parseShort(s);
  String txt="13.5";
  Double x =Double.parseDouble(txt);
  ```

Java.lang.Integer Class

1. static int MAX_VALUE: $2^{31}-1$.

2. static int MIN_VALUE : -2^{31}.

3. Integer(int value), Integer(String s)

4. java.lang.Integer.compare()

5. Integer obj1 = new Integer("25"); Integer obj2 = new Integer("10");

6.   int **retval** = obj1.compareTo(obj2);

7. int **retval** = Integer.compare(obj1,obj2);

8. byte byteValue(), double doubleValue(), float floatValue()…

9. static int parseInt(String s), String toString(), static String toString(int i)

10. boolean equals(Object obj)

Java.lang.Double Class

- Double(double value),
  Double(String s)

- byte byteValue(), double
  doubleValue(), float
  floatValue(), int intValue(), ..

- static int compare(double d1,
  double d2), int compareTo(Double
  anotherDouble)

- boolean **isNaN**(), static double
  parseDouble(String s), String
  toString(), static String
  toString(double d)

java.lang.Character Class

- static char toUpperCase(char ch)
- static char toLowerCase(char ch)
- static String toString(char c)
- String toString()
- static boolean isWhitespace(char ch)
- static boolean isUpperCase(char ch)
- static boolean isLowerCase(char ch)
- static boolean isLetter(char ch)

Strings

● Java uses the **String**, **StringBuffer**, **StringBuilder and** StringTokenizer classes to encapsulate strings of characters (16-bit Unicode).

java.lang.**Object**
    java.lang.**String** (implements java.lang.CharSequence,
                    java.lang.Comparable<T>, java.io.Serializable)
    java.lang.**StringBuffer** (implements java.lang.CharSequence,
            java.io.Serializable)
    java.lang.**StringBuilder** (implements
            java.lang.CharSequence, java.io.Serializable)

java.lang.String Class

- The String class represents character strings. All string literals in Java programs, such as "abc", are implemented as instances of this class.

```
String a = "A String";

String b = "";
```

- Construct a string

```
String c = new String();

String d = new String("Another
  String");

String e = String.valueOf(1.23);

String f = null;
```

String operators

- ## Operator +

  ```
  String a = "This" + " is a " +
    "String";
  //a = "This is a String"
  ```

- ## String with print()

  ```
  System.out.println("answer="+1 + 2 + 3);
  System.out.println("answer="+(1+2+3));
  System.out.println("Number: "+1.45);
  ```

String methods

- compare(), concat(), equals(), split(), length(), replace(), compareTo(), substring(), …
- Example

  - `String name = "Ly Lao Lo";`
  - `name.toLowerCase(); // "ly lao lo"`
  - `name.toUpperCase(); // "LY LAO LO"`
  - `"   Ly Lao Lo ".trim();// "Ly Lao Lo"`
  - `" Ly Lao Lo".indexOf('L'); // 1`
  - `" Ly Lao Lo".indexOf("La");`
  - `"Ly Lao Lo".length(); // 9`
  - `"Ly Lao Lo".charAt(5); // 'o'`
  - `"Ly Lao Lo".substring(5); // "o Lo"`
  - `"Ly Lao Lo".substring(2,5); // " La"`

- int **compareTo**(String anotherString)
- int **compareToIgnoreCase**(String str)
- String **concat**(String str)
- boolean **endsWith**(String suffix)

  ```
  String str = "www.tutorialspoint.com";
  String endstr1 = ".com";
  String endstr2 = ".org";
  boolean retval1 = str.endsWith(endstr1);
  boolean retval2 = str.endsWith(endstr2);
  ```

- boolean **equals**(Object anObject)
- boolean **equalsIgnoreCase**(String anotherString)
- int **indexOf**(String str)
- int **lastIndexOf**(String str)

- ```
  public String[] split(String regex, int
  limit)
  ```
- ```
  public String[] split(String regex)
  ```
- **1.** `String s1="`*Example* method *Split* a Line into *Words*`";`

  ```
  String[] st1=s1.split("\\s+");

  for(String w1:st1){

      System.out.println(w1); }
  ```
- **2.** `String s2=`"Who are you? Are you pretty? some have called you cute while others just call *you pretty*`.";`

  ```
  String[] st2=s2.split("[\\.\\?\\!]");

  for(String w2:st2){

      System.out.println(w2); }
  ```

- `public String` **replaceAll**`(String regex, String replacement)`
- `public String` **replaceFirst**`(String regex, String replacement)`

1. "some have called you cute while others just call *you pretty"* .

   `replaceAll("\\s+"," ");`

2. `String s3=`"Hanoi is famous for numerous rivers,lakes,and mountains alongside and in the surroundings.";
   `System.out.println(s3.`

   `replaceAll("`**`\\,\\s*`**`", ", "));`

3. "my name is khanh my name is *java* ".
   `replaceAll("is","was");`

Input/Output Data

- **import java.util.Scanner;**
- **Scanner input = new Scanner(System.in);**
- public String next()
- public String nextLine()
- public byte nextByte()
- public short nextShort()
- public int nextInt()
- public long nextLong()
- public float nextFloat()
- public double nextDouble()

Problem with .nextLine() in Java

- `System.out.println("Enter numerical value");`
- `int option;`
- `option = input.nextInt();`
  `System.out.println("Enter 1st string");`
  `String string1 = input.nextLine();`
  `System.out.println("Enter 2nd string");`
  `String string2 = input.nextLine();`
- Solution:
  - `int option = input.nextInt();`
  - `input.nextLine(); // Consume newline left-over`
  - `String str1 = input.nextLine();`
  - `int option = Integer.`**`parse`**`Int(input.nextLine());`

Example

```java
/* Write a program that will accept an array of intergers then
   print out entered value and the sum of values
*/
import java.util.Scanner;
public class InputOutputDemo {
    public static void main (String args[])
    {   int a[]; // array of integers
        int n ;  // number of elements of the array
        int i;   // variable for traversing the array
        Scanner sc= new Scanner(System.in); // object for the keyboard
        System.out.print("Enter number of elements: ");
        n = Integer.parseInt(sc.nextLine());
        a = new int[n]; // mem. allocating for elements of the array
        for (i=0;i<n;i++)
        {  System.out.print("Enter the " + (i+1) + "/" + n + " element: ");
            a[i]=Integer.parseInt(sc.nextLine());
        }
        System.out.print("Entered values: ");
        for (i=0;i<n;i++) System.out.format("%5d", a[i]);
        int S=0;
        for (int x: a) S+=x;
        System.out.println("\nSum of values: " + S);
    }
}
```

Refer to Java documentation:
java.lang.String class,
 - the **format** method,
  - format string
for more details

n= sc.nextInt();

Output - Chapter01 (run) #2

```
run:
Enter number of elements: 5
Enter the 1/5 element: 1
Enter the 2/5 element: 4
Enter the 3/5 element: 2
Enter the 4/5 element: 0
Enter the 5/5 element: 7
Entered values:     1    4    2    0    7
Sum of values: 14
BUILD SUCCESSFUL (total time: 11 seconds)
```

Regular Expression in Java

- The `java.util.regex` package primarily consists of three classes: `Pattern`, `Matcher`, and `PatternSyntaxException`.

  - A **Pattern** object is a compiled representation of a regular expression. The Pattern class provides no public constructors. To create a pattern, you must first invoke one of its public static compile methods, which will then return a Pattern object.

  - A **Matcher** object is the engine that interprets the pattern and performs match operations against an input string.

  - A **PatternSyntaxException** object is an unchecked exception that indicates a syntax error in a regular expression pattern.

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match
gi	Perform a global case-insensitive match
^	Get a match at the beginning of a string
$	Get a match at the end of a string
[xyz]	Find any character in the specified character set
[^xyz]	Find any character not in the specified character set
\w	Find any Alphanumeric character including the unserscore
\d	Find any single digit
\s	Find any single space character
?	Find zero or one occurrence of the regular expression
*	Find zero or more occurrence of the regular expression
+	Find one or more occurrence of the regular expression
()	Find the group of characters inside the parentheses & stores the matches string
X{n}	Matches any string that contains a sequence of *n* X's
X{n,m}	Matches any string that contains a sequence of X to Y *n*'s

"^\\(?\\d{3}\\)?-?\\s*\\d{3}\\s*-?\\d{4}$"

```java
public class MatchPhoneNumber {
  public static boolean isPhoneValid(String
  phone){
    boolean retval = false;
    String regex =
    "^\\(?\\d{3}\\)?-?\\s*\\d{3}\\s*-?\\d{4}$";
    retval = phone.matches(regex);
    if (retval)
      System.out.println("MATCH "+phone + "\n");
    return retval; }
  public static void main(String args[]) {
      isPhoneValid("(234)-  765  -8765");
      isPhoneValid("999-585-4009");
      isPhoneValid("1-585-4009");   }
}
```

```java
public class MatchRollNumber {
  public static boolean isRollNumber(String
  id){
    boolean retval = false;
    String regex =
      "^[Bb]{1}\\d{2}[A-Za-z]{4}\\d{3}$";
      //"^[NXnx]{1}\\d{3}[A-Za-z]{4}$"
    retval = id.matches(regex);
    if (retval)
      System.out.println("MATCH "+id+ "\n");
    return retval; }
  public static void main(String args[]) {
      isRollNumber("B13DCCN765");
      isRollNumber(" B13DCCN8584");
      isRollNumber("b12dcat321");   }}
```

StringBuffer, StringBuilder Classes

- Java's **StringBuffer** and **StringBuilder** classes represent strings that can be dynamically modified.
  - StringBuffer is **threadsafe**.
  - StringBuilder (introduced in 5.0) is not threadsafe.
- Almost of their methods are the same as methods in the String class.
- These classes do not use string pool, thus we cannot write StringBuffer t = "ABC";
- We cannot use the operator + to their objects.

Thread: Unit of code (method) is running
Multi-threading program: A program has some threads running concurrently. If 2 threads access common data, their values are not un-predictable. So, in multi-thread programming, JVM supports a mechanism in which accesses to common resources must carry out in sequence based on synchronized methods.
Threadsafe class: A class with synchronized methods.

StringBuilder

public final class **StringBuilder** extends Object
implements Serializable, CharSequence

- The StringBuilder class was introduced in 5.0. It is nearly identical to StringBuffer.

- Major difference: string builders are not threadsafe.

- If you want multiple threads to have concurrent access to a mutable string, use a string buffer.

- If your mutable string will be accessed only by a single thread, there is an advantage to using a string builder, which will generally execute faster than a string buffer.

StringBuilder - Class constructors

Constructor & Description
StringBuilder() This constructs a string builder with no characters in it and an initial capacity of 16 characters.
StringBuilder(int capacity) This constructs a string builder with no characters in it and an initial capacity specified by the capacity argument.
StringBuilder(String str) This constructs a string builder initialized to the contents of the specified string.

StringBuilder - Class methods

StringBuilder append(String str) This method appends the specified string to this character sequence.

StringBuilder append(StringBuffer sb) This method appends the specified StringBuffer to this sequence.

char charAt(int index) This method returns the char value in this sequence at the specified index.

StringBuilder delete(int start, int end) This method removes the characters in a substring of this sequence.

StringBuilder deleteCharAt(int index) This method removes the char at the specified position in this sequence.

int indexOf(String str) This method returns the index within this string of the first occurrence of the specified substring.

int indexOf(String str, int fromIndex) This method returns the index within this string of the first occurrence of the specified substring, starting at the specified index.

int length() This method returns the length (character count).
StringBuilder replace(int start, int end, String str) This method replaces the characters in a substring of this sequence with characters in the specified String.
StringBuilder reverse() This method causes this character sequence to be replaced by the reverse of the sequence.
String substring(int start) This method returns a new String that contains a subsequence of characters currently contained in this character sequence.
String substring(int start, int end) This method returns a new String that contains a subsequence of characters currently contained in this sequence.
String toString() This method returns a string representing the data in this sequence.
StringBuilder insert(int offset, String str) This method inserts the string into this character sequence.

Example

1. ```
StringBuilder s=new
StringBuilder("D14- ");
s.append("CN");//D14- CN
```
2. ```
s.insert(4,"CQ");//D14-CQ CN
```
3. ```
s.delete(3,5);// D14 CN
```
4. ```
s.reverse();
```

Normal Text

```java
public class WrapperDemo {
public static String normalText(String line)
    {
        String out = "";
        line = line.toLowerCase();
        line = line.replaceAll("\\s+", " ");
        line = line.replaceAll(" \\.", "\\.");
        line = line.replaceAll("\\.", "\\. ");
        line = line.replaceAll(" \\,", "\\,");
        line = line.replaceAll("\\,", "\\, ");
        line = line.replaceAll("\\s+", " ");
        line = line.trim();
```

```java
out=line;
boolean isCap = true;
char c;
StringBuilder strb = new StringBuilder("");
for (int i = 0; i < out.length()-1; i++) {
  c = out.charAt(i);
  if (c == '.') {
    isCap = true;
  }
  if (isCap && Character.isAlphabetic(c)) {
    c = Character.toUpperCase(c);
    isCap = false;
  }
  strb.append(c);
  }
```

```java
    out = strb.toString();
    if (out.charAt(out.length()-1) != '.') {
        out = out + ".";
    }
    return out;
    }
public static void main(String[] args) {
    String line="We    were   both   young ,
     when    I first saw you .
     i    close my eyes and the flashback
starts";
System.out.println(normalText(line));
    }
```

Reversible

```
public boolean reversing(int n){
       StringBuilder sn=new
   StringBuilder(Integer.toString(n));
       return
   sn.toString().equals(sn.reverse().to
   String());
     }
```

The *StringBuffer - threadsafe*

public final class **StringBuffer** extends [Object](#)
                        implements [Serializable](#), [CharSequence](#)

```java
public class StringBufferDemo {
 public static void main(String aegs[]){
     StringBuffer sBuf= new StringBuffer ("01234567");
     System.out.println(sBuf);
     sBuf.append("ABC");
     System.out.println(sBuf);
     sBuf.insert(2, "FAT PERSON");
     System.out.println(sBuf);
     sBuf.reverse();
     System.out.println(sBuf);
 }
}
```

```
run:
01234567
01234567ABC
01FAT PERSON234567ABC
CBA765432NOSREP TAF10
```

Constructors

Constructor & Description
StringBuffer()   This constructs a string buffer with no characters in it and an initial capacity of 16 characters.
StringBuffer(int capacity)   This constructs a string buffer with no characters in it and the specified initial capacity.
StringBuffer(String str)   This constructs a string buffer initialized to the contents of the specified string.

- public **synchronized** StringBuffer append(String s)

- public **synchronized** StringBuffer insert(int offset, String s)

- public **synchronized** StringBuffer replace(int startIndex, int endIndex, String str)

- ………..

Example: StringBuffer vs StringBuilder

```java
public class ConcatTest{
  public static void main(String[] args){
    long startTime = System.currentTimeMillis();
    StringBuffer sb1 = new StringBuffer("Java");
    for (int i=0; i<10000; i++){
      sb1.append("D14CN-PTIT"); }
      System.out.println("the total time(StringBuffer): "
  + (System.currentTimeMillis() - startTime) +
  "ms");
      startTime = System.currentTimeMillis();
      StringBuilder sb2 = new StringBuilder("Java");
    for (int i=0; i<10000; i++){
      sb2.append("D14CN-PTIT"); }
    System.out.println("the total time (StringBuilder): "
  + (System.currentTimeMillis() - startTime) +
  "ms");
  } }
```

the total *time*(StringBuffer): 15ms
the total *time*(StringBuilder): 0ms

StringTokenizer Class

- The **java.util.StringTokenizer** class allows you to break a string into tokens. It is simple way to break string.

- Constructors:

  - StringTokenizer(String str): creates StringTokenizer with specified string and delimeter.

  - StringTokenizer(String str, String delim): creates StringTokenizer with specified string and delimeter.

Methods of StringTokenizer class

Public method	Description
boolean hasMoreTokens()	checks if there is more tokens available.
String nextToken()	returns the next token from the StringTokenizer object.
String nextToken(String delim)	returns the next token based on the delimeter.
boolean hasMoreElements()	same as hasMoreTokens() method.
Object nextElement()	same as nextToken() but its return type is Object.
int countTokens()	returns the total number of tokens.

Example

```
import java.util.StringTokenizer;
public class Simple{
public static void main(String
  args[]){
 StringTokenizer st = new
 StringTokenizer("I work at HN.I am
 a lecturer.I love HN.","\\.");
 while(st.hasMoreTokens()) {
 System.out.println(st.nextToken());
 } } }
```

- **By default StringTokenizer breaks String**

```
String str = "I am sample string and will be
   tokenized on space";

StringTokenizer dt=new StringTokenizer(str);

while (dt.hasMoreTokens()){

   System.out.println(dt.nextToken());}
```

- **Multiple delimiters**

```
String s ="Who am i?Lan is my friend.I love
Lan!How Lan love Him? of course i know!";

StringTokenizer mt = new StringTokenizer(s,
".?!");

while (mt.hasMoreTokens()){

   System.out.println(mt.nextToken());

}
```

Summary

- The traditional features of the language, including: variables, arrays, data types, operators, and control flow.

- Wapper class, 4 String classes

- Regular Expression for data validation