CMT205 Object-Oriented Development with Java

Week 2

Arithmetic
Decision
Loop Control
Keyboard Input
Command Line Input
Characters

- **Arithmetic**
- Decision
- Loop Control
- Keyboard Input
- Command Line Input
- Characters

Arithmetic Operators

- Arithmetic operators:

Operator	Meaning
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulus (remainder of integer division)

- Arithmetic operators follow normal precedence, e.g. 6+(3+4)*5

Mathematical Operators Example

- MathOps.java

```
public class MathsOps
{
   public static void main(String[] args)
   {
      int a, b, c, d, e;
      a = 128;
      b = 6;
      c = a * b;
      d = a / b;
      e = a % b;
      System.out.println("a = " + a + "  b = " + b
        + "  c = " + c + "  d = " + d + "  e = " + e);
   }
}
```

Output:

```
a = 128  b = 6  c = 768  d = 21  e = 2
```

Mathematical Methods

- Note: these are static methods of **Math** class.

Method	Operation
Math.sqrt(x)	square root of **x**
Math.pow(x,y)	**x** raised to the power of **y**
Math.sin(x)	sine of **x**
Math.cos(x)	cosine of **x**
Math.tan(x)	tangent of **x**
Math.asin(x)	arc sine of **x**
Math.acos(x)	arc cosine of **x**
Math.atan(x)	arc tangent of **x**
Math.toDegrees(x)	converts **x** radians to degrees
Math.toRadians	converts **x** degrees to radians
Math.exp(x)	**e** to the power of **x**
Math.log(x)	natural log of **x**
Math.round(x)	closest integer to **x**
Math.ceil(x)	smallest integer greater than or equal to **x**
Math.floor(x)	largest integer less than or equal to **x**
Math.abs(x)	absolute value of **x**

Use of methods of the **Math** class

- Contents of **SinePi.java**

```
// Use of sin method and constant PI of the Math class
import java.lang.Math;
public class SinePi
{
   public static void main( String[] args )
   {
      int degrees = 30;
      double radians;
      radians = degrees * 2 * Math.PI / 360;
      System.out.println( Math.sin( radians ) );
   }
}
```

Output:

```
0.49999999999999994
```

Use of methods of the **Math** class

- Contents of **SineMore.java**

```
// Use of methods toRadians and sin of the Math class
import java.lang.Math;
public class SineMore
{
   public static void main( String[] args )
   {
      int degrees = 30;
      double radians = Math.toRadians( degrees );
      System.out.println( Math.sin( radians ) );
   }
}
```

Output:

```
0.4999999999999994
```

Using Different Number Types

- Different number types have different behaviours
- For example, / (division) can be
  - Integer division: if both operands are integers (constants or variables of integer type), so 12/5 will return 2 (rounded down)
  - Floating point division: if either operand is a floating point number (**float** or **double**), then it is treated as a floating point division, so 12.0/5 = 2.4; 12/5.0 = 2.4; 12.0/5.0 = 2.4
  - Note: 12 is an integer (int) constant while 12.0 is a floating point (double) constant

Using Different Number Types

- When mixed types are involved in calculation, Java automatically promote (or convert) the low precision operand to match the high precision one
  - For example, for 3 * 5.2, 3 will be automatically converted to a double number
  - This is **implicit type casting**
- Sometimes, you may want to instruct the compiler to convert the type of a variable; **explicit type casting** is used here:
  - (DataType) (expression)
  - For example, (int)(3 * 5.2) will cast the result of 3*5.2 to an integer number

Contents of MixedNumbers.Java

```
// Calculations involving variables of differing types
public class MixedNumbers
{
   public static void main(String[] args)
   {
      int inum = 27;
      double dnum;
      double decResult;
      int intResult;
      decResult = inum / 5 + 10.5;
      System.out.println( decResult );
      decResult = inum / 5.0 + 10.5;
      System.out.println( decResult );
      decResult = (double) ( inum / 5 + 10.5 );
      System.out.println( decResult );
      decResult = (double) inum / 5 + 10.5;
      System.out.println( decResult );
```

Contents of MixedNumbers.Java (cont.)

```
        dnum = 13.75;
        intResult = (int) dnum * 100;
        System.out.println(intResult);
        intResult = (int) ( dnum * 100 );
        System.out.println( intResult );
        dnum = 4.35;
        intResult = (int) ( dnum * 100 );
        System.out.println( intResult );
        intResult = (int) Math.round( 4.35 * 100 );
        System.out.println( intResult );
    }
}
```

Contents of MixedNumbers.Java (cont.)

- **Output:**

```
15.5
15.9
15.5
15.9
1300
1375
434
435
```

Increment and Decrement Operators

- The **increment** operator (**++**) is used to increase the value of a variable by **1**.

- Similarly, the **decrement** operator (**--**) is used to decrease the value of a variable by **1**.

- **count = count + 1;**
- may be written
- **count++;**

- **count = count – 1;**
- May be written
- **count--;**

Increment and Decrement Operators (cont.)

- ++, -- can be placed before (prefix) or after (postfix) of a variable, e.g. ++count, count++
- No difference if increment/decrement is applied on its own
- Different if this is part of the expression
  - if ++ or – is in the **prefix** position, the increment/decrement is applied before the further calculation
  - if ++ or – is in the **postfix** position, the increment/decrement is applied after the further calculation
  - Example:

```
int count = 3;
int pre = ++count;
int post = count++;
```

Shortcut Arithmetic Operators

- Instead of writing `a = a + 3`, Java allows a shortcut way `a += 3`
- This is more useful when the variable name is longer or more complicated
- All the typical operators allow this:

```
+=      a+=3;    ⇔ a=a+3;
-=      a-=3;    ⇔ a=a-3;
*=      a*=3;    ⇔ a=a*3;
/=      a/=3;    ⇔ a=a/3;
%=      a%=3;    ⇔ a=a%3;
```

- Arithmetic
- **Decision**
- Loop Control
- Keyboard Input
- Command Line Input
- Characters

Comparison Operators

Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

The **if** statement

- Syntax

```
if  (condition)
     statement
```

- Examples

```
if  ( amount <= balance )
     balance = balance - amount;
```

- Several statements may be grouped together to form a ***block statement*** (or ***compound statement***):

```
if  ( amount <= balance )
{
     newBalance = balance - amount;
     balance = newBalance;
}
```

Content of **Even.Java**

```
public class Even
{
   public static void main(String[] args)
   {
      int number = 23;
      if ( ( number % 2 ) == 0 )
         System.out.println( number
                        + " is an even number" );
      else
         System.out.println( number
                        + " is an odd number" );
   }
}
```

Output:

```
23 is an odd number
```

Java Strings

- String literals: a sequence of characters within " "
- In Java, both string literals and variables are instances of the **String** class.
- **String** objects in Java are immutable
  - Every time the string is modified, a new **String** object is returned (and the original one is **unchanged**)

- More in-depth discussion later

Java Strings (cont.)

- **String** class methods
  - **int length()** returns the length of the string
  - **String toUpperCase()** converts every character to upper case
  - **String toLowerCase()** converts every character to lower case
  - **String substring(int beginIndex)** returns a substring that starts from the index of **beginIndex**
  - **String substring(int beginIndex, int endIndex)** returns a substring that starts from **beginIndex** and finishes before **endIndex**
  - **+:** concatenate two strings
  - **equals (equalsIgnoreCase)**: compare if two strings are identical (or identical with case ignored)
- **Note:** == does not compare if two Strings are identical; it only compares if two references are the same (more later)

Content of **StringMethods.java**

```
//
// Use of String methods
//
public class StringMethods
{
   public static void main( String[] args )
   {
      String surName = "Smith";
      String foreName = "John";
      String fullName;
      int numChars;
      fullName = foreName + " " + surName;
      System.out.println(fullName);
      fullName = foreName.toUpperCase() + " "
                                 + surName.toUpperCase();
      System.out.println(fullName);
```

Content of **StringMethods.java** (cont.)

```
        fullName = foreName.toLowerCase() + " "
                                  + surName.toLowerCase();
        System.out.println(fullName);
        numChars = surName.length();
        System.out.println("The length of \"" + surName
                 + "\" is " + numChars + " characters");
        fullName = foreName + " " + surName;
        surName = fullName.substring(5);
        System.out.println("The surname is " + surName);
        foreName = fullName.substring(0,4);
        System.out.println("The forename is "+ foreName);
    }
}
```

Example: **StringMethods.java** (cont.)

- Output

```
John Smith
JOHN SMITH
john smith
The length of "Smith" is 5 characters
The surname is Smith
The forename is John
```

Contents of **CompareStrings.java**

```
// Comparison of strings
public class CompareStrings
{
   public static void main( String[] args )
   {
      String firstString = "Hello";
      String secondString = "HELLO";
      boolean equalString;
      equalString = firstString.equals( secondString );
      if ( equalString == true )
         System.out.println("Strings are identical");
      else
         System.out.println("Strings are different");
      equalString = firstString.equalsIgnoreCase( secondString );
      if ( ! equalString )
         System.out.println("Strings are different");
      else
         System.out.println("Strings are identical");
   }
}
```

Output:

```
Strings are different
Strings are identical
```

Multiple if

- There are cases where multiple conditions need to be checked.
- The statement can as well be **if** statements.
- Exercise: if the average mark (double variable **avgMark**) is larger than or equal to 70, the degree (String variable **result**) is **“distinct”**; otherwise, if the average mark is between 50 (inclusive) and 70 (exclusive), the degree is “**pass”**. Less than 50 means **“fail”**.

Contents of **MultipleIf.java**

```
public class MultipleIf
{
   // Use of multiple if statements
   public static void main( String[] args )
   {
      int digit = 3;
      String digitName;
      if ( digit == 0 ) digitName = "zero";
      else if ( digit == 1 ) digitName = "one";
      else if ( digit == 2 ) digitName = "two";
      else if ( digit == 3 ) digitName = "three";
      else if ( digit == 4 ) digitName = "four";
      else if ( digit == 5 ) digitName = "five";
      else if ( digit == 6 ) digitName = "six";
      else if ( digit == 7 ) digitName = "seven";
      else if ( digit == 8 ) digitName = "eight";
      else if ( digit == 9 ) digitName = "nine";
      else digitName = "";
      System.out.println( digitName );
   }
}
```

Output:

```
three
```

The **switch** statement

- **switch** is an alternative to **if**, useful when there are multiple paths of execution.
- You **typically** use a ***switch*** to branch on ***integer*** values. You ***cannot*** use a ***switch*** statement to branch on ***floating-point*** values.
- For Java SE 7 or later, you can use a **String** object in the **switch** statement as well.

```
switch ( intVariable )
{
     case intValue1:  . . .  ; break;
     case intValue2:  . . .  ; break;
       .
       .
     default:  . . .  ; break;
}
```

The **switch** statement (cont.)

- **switch** explained:
  - **break** is needed. Otherwise the program will run to the statement for the following case branch.
  - **default** is used when no case condition is matched
- Differences from **if**
  - Compare multiple possible values
  - Only == condition can be used

Switch Example

```
public class Switch
{
   // Use of switch statement
   public static void main( String[] args )
   {
      int digit = 9;
      String digitName;
      switch ( digit )
      {
         case 0:  digitName = "zero"; break;
         case 1:  digitName = "one"; break;
         case 2:  digitName = "two"; break;
         case 3:  digitName = "three"; break;
         case 4:  digitName = "four"; break;
         case 5:  digitName = "five"; break;
         case 6:  digitName = "six"; break;
         case 7:  digitName = "seven"; break;
         case 8:  digitName = "eight"; break;
         case 9:  digitName = "nine"; break;
         default: digitName = ""; break;
      }
      System.out.println( digitName );
   }
}
```

Output:

```
nine
```

Switch String Example

```
public class SwitchString
{
   // Use of switch statement for Strings
   // Java SE 7 or later
   public static void main( String[] args )
   {
      String digitName = "nine";
      int digit;
      switch ( digitName )
      {
         case "zero":  digit = 0; break;
         case "one":   digit = 1; break;
         case "two":   digit = 2; break;
         case "three": digit = 3; break;
         case "four":  digit = 4; break;
         case "five":  digit = 5; break;
         case "six":   digit = 6; break;
         case "seven": digit = 7; break;
         case "eight": digit = 8; break;
         case "nine":  digit = 9; break;
         default: digit = -1; break;
      }
      System.out.println( digit );
   }
}
```

Output:

9

Boolean Expressions

- In ***Java***, the value of a ***relational*** expression is either ***true*** or ***false***.
- For example, if an ***integer*** variable ***x*** contains the value ***9***, then the value of ***x < 10*** is ***true***.
- Primitive type **boolean** is defined to hold either **true** or **false**.

Contents of **StoreBoolean.Java**

```
public class StoreBoolean
{
   // Use of boolean variables
   public static void main( String[] args )
   {
      int number = 2;
      boolean state;
      if ( number < 2 )
         state = true;
      else
         state = false;
      System.out.println( state );
      if ( ! ( number < 2 ) )
         state = true;
      else
         state = false;
      System.out.println( state );
   }
}
```

Output:

```
false
true
```

Boolean Expressions

- Booleans store true/false values
- The following block of code

```
if ( number < 2 )
          state = true;
       else
          state = false;
```

is equivalent to

```
state = number < 2;
```

Selection Operator

- ***Java*** has a ***selection*** operator of the form:-

test ? value1 : value2;

- The statement

```
y = x >= 0 ? x : -x;
```

is shorthand for

```
if   ( x >= 0 )
     y = x;
else
     y = -x;
```

Selection Operator (cont.)

test ? value1 : value2;

- The ***selection*** operator combines ***expressions*** and yields another ***expression***.
- Expressions have values
  - If the **test** is true, the value is **value1**
  - If the **test** is false, the value is **value2**

- The ***if/else statement*** combines ***statements*** and yields another ***statement***. ***Statements*** do not have ***values***.

Absolute Value using Selection

```
public class Absolute
{
   // Use of selection operator
   public static void main( String[] args )
   {
      int x = 5;
      int y = -10;
      int z;
      System.out.println( "Value of x is " + x );
      System.out.println( "Value of y is " + y );
      z = x >= 0 ? x : -x;
      System.out.println( "Value of z is " + z );
      z = y >= 0 ? y : -y;
      System.out.println( "Value of z is " + z );
   }
}
```

Output:

```
Value of x is 5
Value of y is -10
Value of z is 5
Value of z is 10
```

Relational Operators

- Multiple relational expressions can be combined
- For example, the degree is "pass" if avgMark >= 50 and avgMark < 70.
- Logical operators are useful for this purpose
  - **&&** logical and operator
  - **||** logical or operator
  - **!** logical not operator

&& (and) Operator

- If **A** and **B** are ***relational expressions***, the truth table for the ***&& logical operator*** is

A	B	A && B
false	false	false
false	true	false
true	false	false
true	true	true

|| (or) Operator

- If **A** and **B** are ***relational expressions***, the truth table for the **||** ***logical operator*** is

A	B	A || B
False	false	false
false	true	true
true	false	true
true	true	true

! (not) Operator

- If **A** is a ***relational expression***, the truth table for the **! (not)** ***logical operator*** is

A	! A
true	**false**
false	**true**

Boolean Evaluation Shortcut

- Assume **A** and **B** are two individual tests.
- If **A** is **false**, **A && B** must be **false**, which java can (and does) decide without computing **B**.
- If **A** is **true**, **A || B** must be **true**, which java can (and does) decide without computing **B**.
- Application: `if (x >= 0 && Math.sqrt(x) > 2.0)` **…** The shortcut is
  - More efficient
  - Avoids possible computation errors
  - Note that this is different from

```
if (Math.sqrt(x) > 2.0 && x >= 0) …
```

- Arithmetic
- Decision
- **Loop Control**
- Keyboard Input
- Command Line Input
- Characters

Loops

- Many problems can be solved by repeating similar operations, e.g.
  - Finding the maximum number (or the average) of a sequence (e.g. finding the best score/average score)
  - Process all the student records (e.g. print all the relevant records with a query)
  - Stylise an image by modifying each pixel
  - Draw a 3D scene by painting lots of elements (triangles)

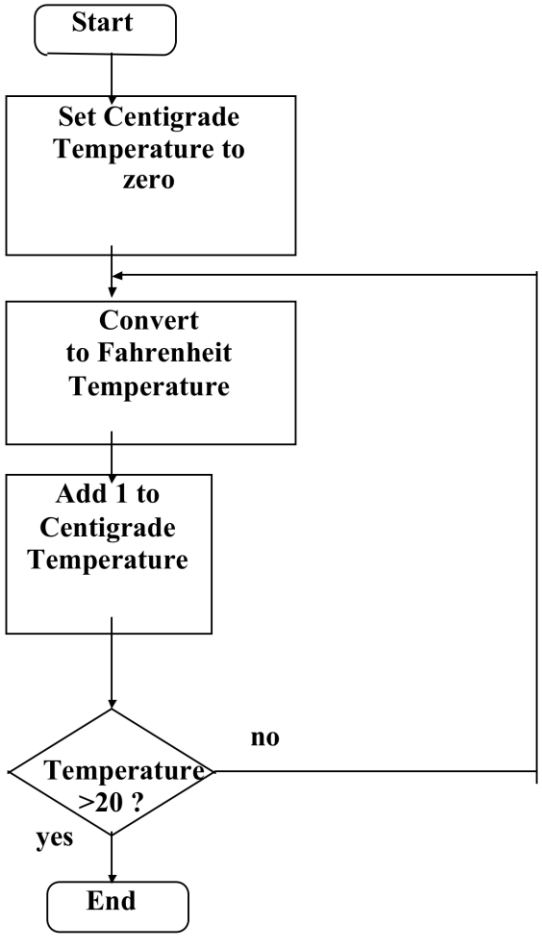
Loops

- Java provides the following language constructs to support loops
  - **while**
  - **do … while**
  - **for**

A Temperature Example

- Convert 0, 1, 2, …, 20 Centigrade to corresponding Fahrenheit temperature

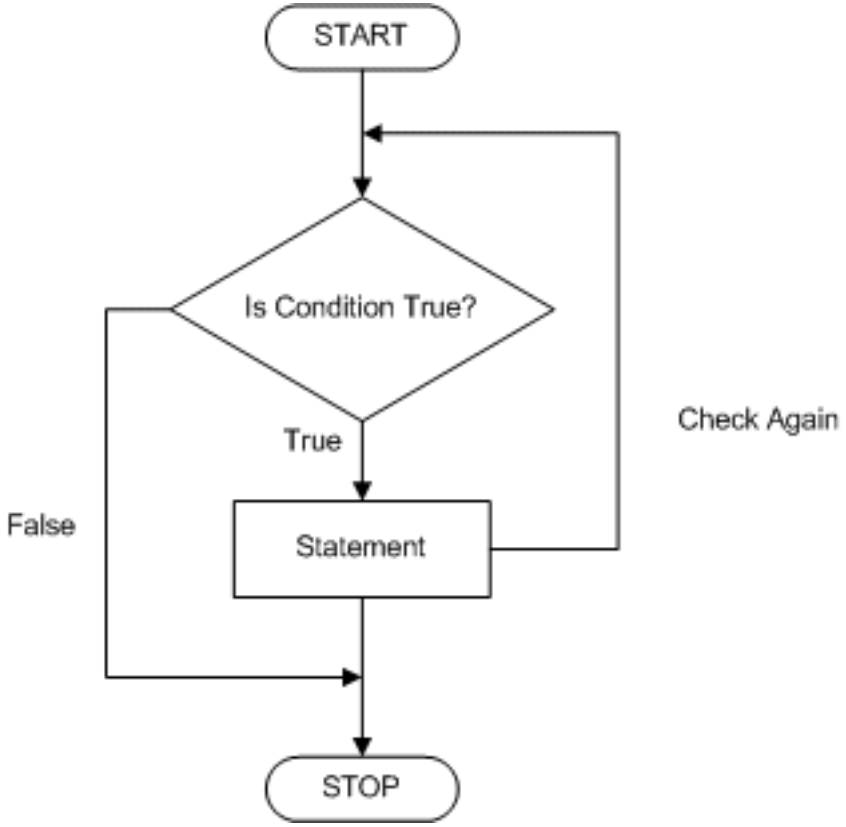
- Use a flow-chart or pseudecode to help design the algorithm



The **while** statement

```
while (condition)
    Statement
```

- The statement above can be a compound statement (by using {})
- The condition is checked before running the statement



Contents of **WhileLoop.java**

```
public class WhileLoop
{
   // Temperature conversion using while loop
   public static void main(String[] args)
   {
      int centTemp, fahTemp;
      centTemp = 0;
      while ( centTemp <= 20 )
      {
         fahTemp = ( centTemp * 9 ) / 5 + 32;
         System.out.println( centTemp
             + " degrees C = "
             + fahTemp + " degrees F" );
         centTemp++;
      }
   }
}
```

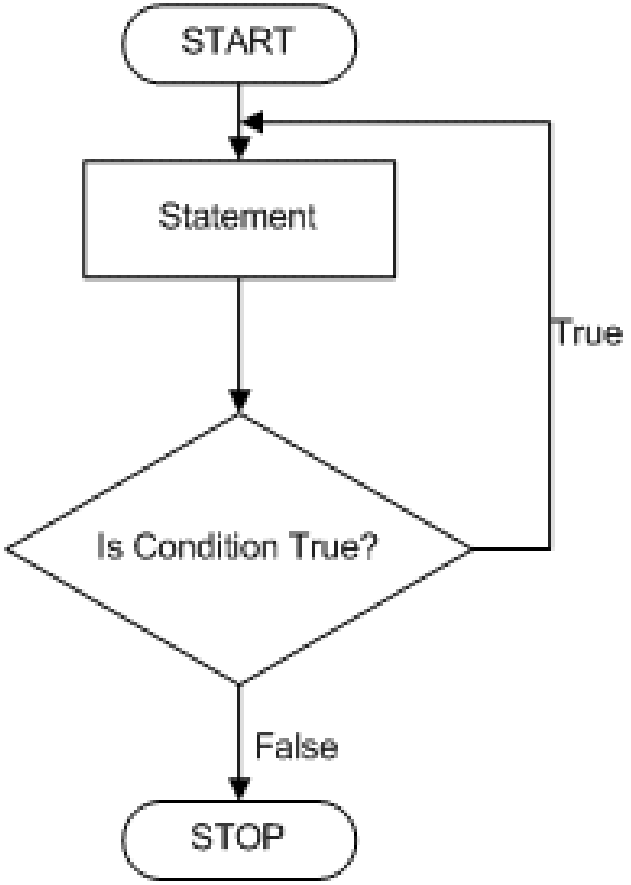
WhileLoop.java Output

```
0 degrees C = 32 degrees F
1 degrees C = 33 degrees F
2 degrees C = 35 degrees F
3 degrees C = 37 degrees F
4 degrees C = 39 degrees F
5 degrees C = 41 degrees F
6 degrees C = 42 degrees F
7 degrees C = 44 degrees F
8 degrees C = 46 degrees F
9 degrees C = 48 degrees F
10 degrees C = 50 degrees F
11 degrees C = 51 degrees F
12 degrees C = 53 degrees F
13 degrees C = 55 degrees F
14 degrees C = 57 degrees F
15 degrees C = 59 degrees F
16 degrees C = 60 degrees F
17 degrees C = 62 degrees F
18 degrees C = 64 degrees F
19 degrees C = 66 degrees F
20 degrees C = 68 degrees F
```

The **do … while** statement

```
do
        Statement
while (condition)
```

- The statement will be run once even if the condition is false in the first place



Contents of **DoLoop.java**

```
public class DoLoop
{
   // Temperature conversion using do loop
   public static void main(String[] args)
   {
      int centTemp, fahTemp;
      centTemp = 0;
      do
      {
         fahTemp = ( centTemp * 9 ) / 5 + 32;
         System.out.println( centTemp
                     + " degrees C = "
                     + fahTemp + " degrees F" );
         centTemp++;
      } while ( centTemp <= 20 );
   }
}
```

DoLoop.java Output

```
0 degrees C = 32 degrees F
1 degrees C = 33 degrees F
2 degrees C = 35 degrees F
3 degrees C = 37 degrees F
4 degrees C = 39 degrees F
5 degrees C = 41 degrees F
6 degrees C = 42 degrees F
7 degrees C = 44 degrees F
8 degrees C = 46 degrees F
9 degrees C = 48 degrees F
10 degrees C = 50 degrees F
11 degrees C = 51 degrees F
12 degrees C = 53 degrees F
13 degrees C = 55 degrees F
14 degrees C = 57 degrees F
15 degrees C = 59 degrees F
16 degrees C = 60 degrees F
17 degrees C = 62 degrees F
18 degrees C = 64 degrees F
19 degrees C = 66 degrees F
20 degrees C = 68 degrees F
```

The **for** statement

```
for (initialisation; condition; update)
   Statement
```

This is equivalent to:

```
   initialisation;
   while (condition)
   {
      Statement
      update
   }
```

Contents of **ForLoop.java**

```
public class ForLoop
{
   // Temperature conversion using for loop
   public static void main(String[] args)
   {
      int centTemp, fahTemp;
      for ( centTemp = 0; centTemp <= 20; centTemp++ )
      {
         fahTemp = ( centTemp * 9 ) / 5 + 32;
         System.out.println( centTemp
                        + " degrees C = "
                        + fahTemp + " degrees F" );
      }
   }
}
```

ForLoop.java Output

```
0 degrees C = 32 degrees F
1 degrees C = 33 degrees F
2 degrees C = 35 degrees F
3 degrees C = 37 degrees F
4 degrees C = 39 degrees F
5 degrees C = 41 degrees F
6 degrees C = 42 degrees F
7 degrees C = 44 degrees F
8 degrees C = 46 degrees F
9 degrees C = 48 degrees F
10 degrees C = 50 degrees F
11 degrees C = 51 degrees F
12 degrees C = 53 degrees F
13 degrees C = 55 degrees F
14 degrees C = 57 degrees F
15 degrees C = 59 degrees F
16 degrees C = 60 degrees F
17 degrees C = 62 degrees F
18 degrees C = 64 degrees F
19 degrees C = 66 degrees F
20 degrees C = 68 degrees F
```

Formatting Integer Output using Spaces

- **Contents of ForLoop.java**

```
// Program to produce a formatted output
public class ForLoop
{
   public static void main( String[] args )
   {
      int centTemp, fahTemp;
      String outputString;
      int itemLength;
      int count;
      for ( centTemp = 0; centTemp <= 20; centTemp++ )
      {
         fahTemp = ( centTemp * 9 ) / 5 + 32;
         // Output the Centigrade temperature to the screen
         // using a field width of 2 characters
         outputString = Integer.toString( centTemp );
         itemLength = outputString.length();
         if ( itemLength < 2 )
         {
            for( count = 1; count <= 2 - itemLength; count++ )
               System.out.print(" ");
         }
```

Formatting Integer Output using Spaces (cont.)

```
            System.out.print( outputString );
            System.out.print(" degrees C = ");
            // Output the Fahrenheit temperature to the screen
            // using a field width of 2 characters
            outputString = Integer.toString( fahTemp );
            itemLength = outputString.length();
            if ( itemLength < 2 )
            {
               for( count = 1; count <= 2 - itemLength; count++ )
                  System.out.print(" ");
            }
            System.out.print( outputString );
            System.out.println(" degrees F");
         }
      }
   }
```

Formatting Integer Output by Defining a Method

```
public class ForLoop
{
   // Method to output an integer to the screen using a given field width
   public static void displayInt( int number, int width )
   {
      String outputString;
      int itemLength, count;
      outputString = Integer.toString( number );
      itemLength = outputString.length();
      if ( itemLength < width )
         for ( count = 1; count <= width - itemLength; count++ )
            System.out.print( " " );
      System.out.print( outputString );
   }
   public static void main( String[] args )
   {
      int centTemp, fahTemp;
      for ( centTemp = 0; centTemp <= 20; centTemp++ )
      {
         fahTemp = ( centTemp * 9 ) / 5 + 32;
               displayInt( centTemp, 2 );
         System.out.print( " degrees C = " );
               displayInt( fahTemp, 2 );
         System.out.println( " degrees F" );
      }
   }
}
```

ForLoop.java Formatted Output

```
 0 degrees C = 32 degrees F
 1 degrees C = 33 degrees F
 2 degrees C = 35 degrees F
 3 degrees C = 37 degrees F
 4 degrees C = 39 degrees F
 5 degrees C = 41 degrees F
 6 degrees C = 42 degrees F
 7 degrees C = 44 degrees F
 8 degrees C = 46 degrees F
 9 degrees C = 48 degrees F
10 degrees C = 50 degrees F
11 degrees C = 51 degrees F
12 degrees C = 53 degrees F
13 degrees C = 55 degrees F
14 degrees C = 57 degrees F
15 degrees C = 59 degrees F
16 degrees C = 60 degrees F
17 degrees C = 62 degrees F
18 degrees C = 64 degrees F
19 degrees C = 66 degrees F
20 degrees C = 68 degrees F
```

Decimal Places

- The **NumberFormat** class in the **java.text** package allows the value stored in a **double** to be printed to a given number of decimal places.
  - First call the **static** method **getNumberInstance** of the **NumberFormat** class to obtain a reference to a general purpose number format for the current default locale.
  - Then, **setMaximumFractionDigits** method of the **NumberFormat** object sets the maximum number of fraction digits.
  - Similarly, **setMinimumFractionDigits** method of the **NumberFormat** object sets the minimum number of fraction digits.

Decimal Places (cont.)

- For example, set the maximum number of fraction digits to **2**, numbers are rounded to (up to) two fraction digits.
- So **0.2875** will be converted to the string **0.29.**
- **0.2975** will be converted to the string **0.3** and not **0.30**.
- If the minimum number of fraction digits is also set to **2**, then the trailing zeros will be preserved (e.g. for currency).

Contents of **Decimals.java**

```
import java.text.NumberFormat;
public class Decimals
{
   // Division of numbers 1 to 8 by 8
   public static void main(String[] args)
   {
      // Display numbers with two decimal places
      NumberFormat formatter = NumberFormat.getNumberInstance();
      formatter.setMinimumFractionDigits( 2 );
      formatter.setMaximumFractionDigits( 2 );
      double number, result;
      for ( number = 1; number <= 8; number++ )
      {
         result = number / 8;
         System.out.println( formatter.format( result ) );
      }
   }
}
```

Decimals.java Output

```
0.12
0.25
0.38
0.50
0.62
0.75
0.88
1.00
```

Using the **PrintStream** class

- A **PrintStream** adds functionality to another output stream, i.e. the ability to print representations of various data values conveniently.
- Unlike other output streams, a **PrintStream** never throws an **IOException** (detail coming soon); instead, exceptional situations merely set an internal flag that can be tested via the **checkError** method

Using the **PrintStream** class (cont.)

- PrintStream.format(*String format*, …)
  - Write a formatted string to the output stream
- PrintStream.printf(*String format*, …)
  - The same as PrintStream.format
- PrintStream.print (variable)
  - Print the variable to the output stream
- PrintStream.println(variable)
  - Print the variable to the output stream with a carriage return

Using the **PrintStream** class (cont.)

- Format String
  - A normal string
  - With format specifiers, replaced by actual variables in the following variable list
  - Examples
    - %7d 7 spaces right justified integer
    - %-5d 5 spaces left justified integer
    - %7s 7 spaces right justified string
    - %-7.2f 7 spaces 2 decimals left justified floating point
    - %.2f 2 decimals right justified floating point number

Contents of **FormatOutput.java**

```
import java.io.PrintStream;
public class FormatOutput
{
   public static void main(String[] args)
   {
      PrintStream output = new PrintStream( System.out );
      String str = "Output";
      int inum=27;
      double dnum = 13.75;
      // output right justified values
      output.format("%7s%7d%7.2f\n",  str, inum, dnum );
      // output left justified values
      output.format("%-7s%-7d%-7.2f\n", str, inum, dnum );
      // output values with no leading spaces
      output.format("String is %s", str );
      // calling the printf method is the same as calling the format method
      output.printf("  Integer is %d", inum );
      output.printf("  Real Number is %.2f\n", dnum );
      // check no error has occurred while using PrintStream methods
      output.println( output.checkError() );
   }
}
```

FormatOutput.java Output

```
 Output      27  13.75
Output 27     13.75
String is Output  Integer is 27  Real Number is 13.75
false
```

Constant

- A **constant** is a variable which does not change.
- Constant names typically contain uppercase characters with an occasional underscore.
- A **constant** is declared **final** so that its value cannot be changed once it has been initialised.

Content of **GasBill.java**

```
public class GasBill
{
   // Gas bill calculation
   public static void main( String[] args )
   {
      final double STANDING_CHARGE = 9.56;
      final double COST_OF_UNIT;
      double total;
      COST_OF_UNIT = 0.48;
      int units = 10;
      total = STANDING_CHARGE + COST_OF_UNIT * units;
      System.out.println( "Total gas bill is " + total );
   }
}
```

Output:

```
Total gas bill is 14.36
```

Nested Loops

```
// Example using two for loops
public class Tables
{
   public static void main(String[] args)
   {
      int number1, number2, result;
      for ( number1 = 2; number1 <= 4; number1++ )
      {
          for ( number2 = 1; number2 <= 10; number2++ )
         {
            result = number1 * number2;
            System.out.println( number1 + " X "
                        + number2 + " = " + result );
         }
         System.out.println( "" );
      }
   }
}
```

Tables.java Output

```
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
2 X 6 = 12
2 X 7 = 14
2 X 8 = 16
2 X 9 = 18
2 X 10 = 20

3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
3 X 4 = 12
3 X 5 = 15
3 X 6 = 18
3 X 7 = 21
3 X 8 = 24
3 X 9 = 27
3 X 10 = 30

4 X 1 = 4
4 X 2 = 8
4 X 3 = 12
4 X 4 = 16
4 X 5 = 20
4 X 6 = 24
4 X 7 = 28
4 X 8 = 32
4 X 9 = 36
4 X 10 = 40
```

Working with Loops

- Breaking out a loop using **break**
- Example (**BreakTest.java**):

```
for(int a = 1; a < 10; a++)
{
  System.out.print(a + " ");
  if(a == 5)
      break;
}
System.out.println("You have exited the loop");
```

- Output:

```
1 2 3 4 5 You have exited the loop
```

Working with Loops

- Continuing a loop using **continue**
- Example:

```
for (int a = 1; a < 10; a++)
{
    if(a == 5)
        continue;
    System.out.print(a + " ");
}
```

- Output:

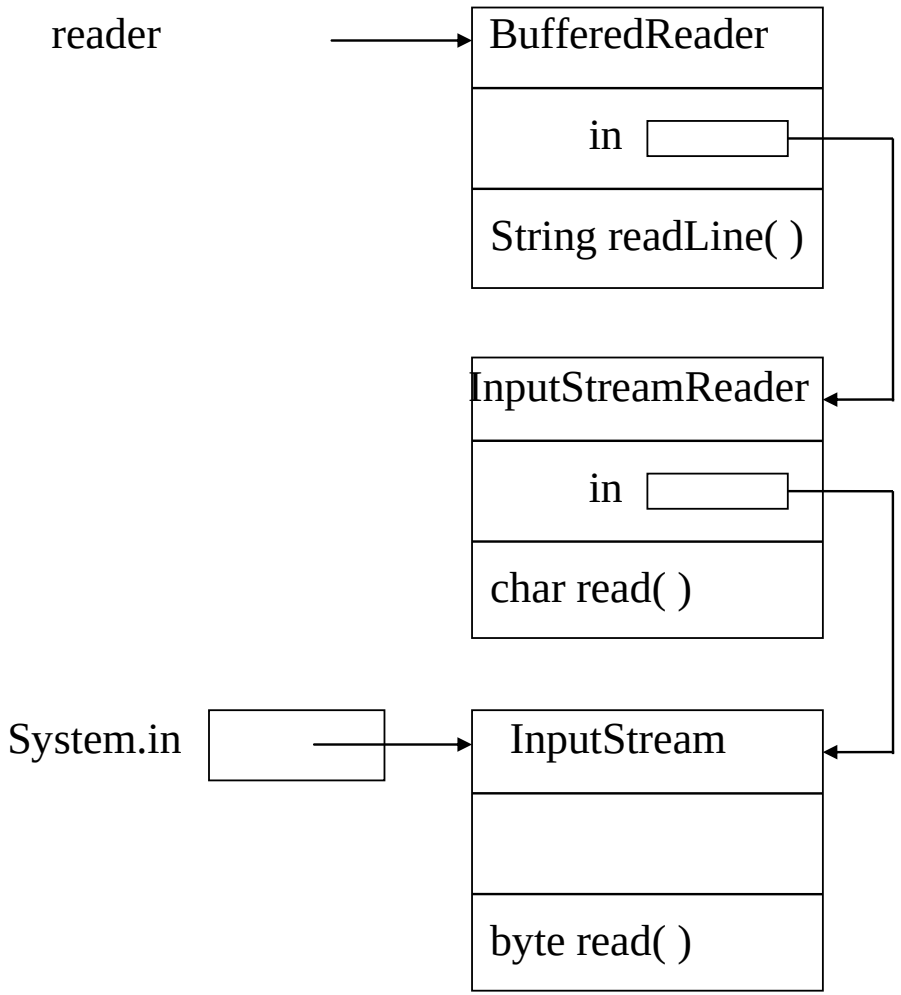
```
1 2 3 4 6 7 8 9
```

- Arithmetic
- Decision
- Loop Control
- **Keyboard Input**
- Command Line Input
- Characters

Reading Input in Java

- InputStream
  - Objects of the ***InputStream*** class read ***bytes***.
  - `System.in` is an example of InputStream.
- InputStreamReader
  - ***Java*** uses ***Unicode*** to represent ***characters*** where each ***character*** is usually made up of ***two bytes***.
  - It is necessary to turn the ***InputStream*** object into a ***Reader*** object, which reads ***characters***, using the ***InputStreamReader*** class.
- BufferedReader
  - To read a ***line*** of input instead of single ***characters***
- If the method for reading a line encounters an error, an ***exception*** is generated which must be caught.

Turning **System.in** into a **BufferedReader** object



Contents of **SimpleReader.java**

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
public class SimpleReader
{
   public static void main(String[] args) throws IOException
   {
      InputStreamReader input = new InputStreamReader( System.in );
      BufferedReader reader = new BufferedReader( input );
      String name = "";
      System.out.print( "Name: " );
      name = reader.readLine();
      System.out.println( "Name entered was '" + name + "'" );
      System.exit( 0 );
   }
}
```

Contents of **Reader.java**

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
public class Reader
{
   public static void main(String[] args) throws IOException
   {
      BufferedReader reader = new BufferedReader(
                  new InputStreamReader( System.in ) );
      String name = "";
      System.out.print( "Name: " );
      name = reader.readLine();
      System.out.println( "Name entered was '" + name + "'" );
      System.exit( 0 );
   }
}
```

Example Output (entered text underlined):

```
Name: James
Name entered was 'James'
```

Exception Handling

- If things go **wrong** in a method, it can (and should) **throw** an **exception** to indicate this. The calling method can then try to take appropriate action.
- If a method (or one of the methods it calls) may throw one or more **exceptions**, they should be **listed** in the **method header**:

  throws ExceptionClassList

e.g.

```
public static void main(String[] args)
                        throws IOException
```

Using **Exception** Handlers

- If a method you call is defined as ***throwing*** an ***exception***, your calling code generally should ***catch*** the exception, and take some appropriate action.
  - The code that may generate an ***exception*** is enclosed in a ***try*** block
  - This is immediately followed by one or more **catch** blocks
  - Each ***catch*** block specifies the type of ***exception*** it can ***catch*** and contains an ***exception*** handler (what the program should do when something unexpected happens)

Using **Exception** Handlers (cont.)

```
try
{
   statement
   statement
   . . .
}
catch ( ExceptionClass exceptionObject )
{
   statement
   statement
   . . .
}
```

More details later

Contents of **Input.java**

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
public class Input
{
   public static void main(String[] args)
   {
      BufferedReader reader = new BufferedReader(new InputStreamReader( System.in ) );
      String name = "";
      try
      {
         System.out.print( "Name: " );
         name = reader.readLine();
      }
      catch ( IOException ioe )
      {
         System.out.println( ioe );
         System.exit( 1 );
      }
      System.out.println( "Name entered was '" + name + "'" );
      System.exit( 0 );
   }
}
```

Create Reusable **KeyboardReader** Class

```
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;

public class KeyboardReader
{
   // constructor for KeyboardReader
   public KeyboardReader( InputStream inStream )
   {
      InputStreamReader input =
                        new InputStreamReader( inStream );
      reader = new BufferedReader( input );
   }
```

Create Reusable **KeyboardReader** Class (cont.)

```
   // method to read a line from the input stream
   public String readLine()
   {
      String inputLine = "";
      try
      {
         inputLine = reader.readLine();
      }
      catch ( IOException ioe )
      {
         System.out.println( ioe );
         System.exit( 1 );
      }
      return inputLine;
   }
   // instance variable
   private BufferedReader reader;
}
```

Repeated Input

- Combine keyboard input with a loop
- Terminate when certain conditions are satisfied, e.g. end-of-input, using a sentinel (impossible input)

Contents of **Mean.java**

```
public class Mean
{
   public static void main( String[] args )
   {
      KeyboardReader keyboard =
                          new KeyboardReader( System.in );
      int sum = 0;
      int count = 0;
      // compute sum of marks
      System.out.println("Enter marks");
      boolean done = false;
      while ( ! done )
      {
         String inputLine = keyboard.readLine();
         if ( inputLine == null )
                done = true;
         else
         {
```

Contents of **Mean.java** (cont.)

```
            try
            {
              int mark = Integer.parseInt( inputLine );
              sum = sum + mark;
              count++;
            }
            catch ( NumberFormatException ne )
            {
               System.out.println( "Not a number!" );
               System.exit( 2 );
            }
        }
    }
    // compute average
    if ( count == 0 )
       System.out.println( "No marks entered" );
    else
       System.out.println( "Average mark is " + sum / count );
  }
}
```

Mean.java Example Output

```
Enter marks
60
70
78
56
^Z
Average mark is 66
```

- Note: inputLine() returns null when the end of input is reached.
  - Ctrl + D on Mac/Linux
  - Ctrl + Z on Windows

Contents of **Sentinel1.java**

```
public class Sentinel1
{
   public static void main( String[] args )
   {
      KeyboardReader keyboard =
                          new KeyboardReader( System.in );
      int sum = 0;
      int count = 0;
      // compute sum of marks
      System.out.println(
            "Enter marks (negative number to finish)" );
      boolean done = false;
      while ( ! done )
      {
         String inputLine = keyboard.readLine();
         int mark = Integer.parseInt( inputLine );
```

Contents of **Sentinel1.java** (cont.)

```
            if ( mark < 0 )
                  done = true;
            else
            {
                  sum = sum + mark;
                  count++;
               }
         }
         // compute average
         if ( count == 0 )
            System.out.println( "No marks entered" );
         else
            System.out.println( "Average mark is "
                                             + sum / count );
      }
}
```

Sentinel1.java Example Output

```
Enter marks (negative number to finish)
60
70
78
56
-1
Average mark is 66
```

Contents of **Sentinel1.java** (cont.)

- In this example
  - `Integer.parseInt()` converts a string to an integer
  - Use a sentinel (impossible input) to terminate the loop (negative number in this case)

Contents of **Sentinel2.java**

```
public class Sentinel2
{
   public static void main( String[] args )
   {
      KeyboardReader keyboard =
                          new KeyboardReader( System.in );
      int sum = 0;
      int count = 0;
      // compute sum of marks
      System.out.println( "Enter marks (Q to finish)" );
      boolean done = false;
      while ( ! done )
      {
         String inputLine = keyboard.readLine();
         if ( inputLine.equalsIgnoreCase( "Q" ) )
               done = true;
```

Contents of **Sentinel2.java** (cont.)

```
        else
        {
            int mark = Integer.parseInt(inputLine);
            sum = sum + mark;
            count++;
        }
    }
    // compute average
    if ( count == 0 )
        System.out.println( "No marks entered" );
    else
        System.out.println( "Average mark is "
                                        + sum / count );
  }
}
```

In this example, "Q" is used as sentinel for terminating the program.

Sentinel2.java Example Output

```
Enter marks (Q to finish)
60
70
78
56
Q
Average mark is 66
```

Use of **StringTokenizer** Class

- The line read in may contain multiple "words" or tokens, separated by **delimiters**
- By default, whitespaces (space character, new line, carriage return, tab) are used as delimiters
- Tokens are contiguous sequence of characters separated by one or more delimiters
- Using **StringTokenizer** class
  - First, create an instance using the string as input
  - countTokens() method returns the number of tokens
  - hasMoreTokens() method checks if more tokens are available
  - nextToken() returns the next token as a String

Contents of **Items.java**

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
import java.util.StringTokenizer;

public class Items
{
   public static void main( String[] args )
   {
      BufferedReader reader = new BufferedReader(
                  new InputStreamReader( System.in ) );
      String inputLine = "";
      try
      {
         System.out.print( "Enter a line of input> " );
         inputLine = reader.readLine();
      }
```

Contents of **Items.java** (cont.)

```
      catch( IOException ioe )
      {
         System.out.println( "I/O error" );
         System.exit( 1 );
      }
      // determine number of items in line
      StringTokenizer tokenizer = new StringTokenizer( inputLine );
      int count = tokenizer.countTokens();
      System.out.println( "There are " + count + " items" );
      // break up line into items
      while ( tokenizer.hasMoreTokens() )
      {
         String item = tokenizer.nextToken();
         System.out.println( item );
      }
      System.exit( 0 );
   }
}
```

Items.java Example Output

```
Enter a line of input> Java is a computer programming language
There are 6 items
Java
is
a
computer
programming
language
```

Use of **Scanner** Class

- **Scanner** object breaks its input into **tokens** using a **delimiter** pattern which by default matches whitespace
- Scanner objects can be used for processing
  - Keyboard input, e.g. **System.in**
  - File input (see later)
  - String
- The resulting **tokens** may then be converted into **values** of different **types** using the various **next** methods
  - hasNext() method: returns true if the next token exists
  - next() method: returns the next token
  - hasNextInt() method: returns true if the next token exists and can be interpreted as an integer
  - nextInt() method: returns the next token as an integer
  - …
  - close() method: closes the Scanner and allows Java to reclaim the Scanner's memory. Use it when the Scanner is no longer needed.

Contents of **StringInput.java**

```
import java.util.Scanner;
public class StringInput
{
   public static void main( String[] args )
   {
      // set up keyboard input
      Scanner in = new Scanner( System.in );
      // read lines until EOF
      while ( in.hasNextLine() )
      {
         String line = in.nextLine();
         if (line.equals("")) break;
         System.out.println( line );
         // split line into tokens
         Scanner elements = new Scanner( line );
         // display each token in line
         while ( elements.hasNext() )
         {
            String str = elements.next();
            System.out.println( str );
         }
         elements.close();
      }
      in.close();
   }
}
```

StringInput.java Example Output

```
Hello, World
Hello, World
Hello,
World
Java is a programming language
Java is a programming language
Java
is
a
programming
language
<ENTER>
```

Note:

End of Input is natural for file or String as input. For keyboard input, Ctrl+Z (on Windows), or Ctrl+D (Linux/Mac) is considered as End of Input.

Empty string is used as a sentinel to indicate no further text is to be entered.

Contents of **IntegerInput.java**

```
import java.util.Scanner;
public class IntegerInput
{
   public static void main( String[] args )
   {
      Scanner in = new Scanner( System.in );
      while ( in.hasNextLine() )
      {
         String line = in.nextLine();
         if (line.equals("")) break;
         System.out.println( line );
         Scanner elements = new Scanner( line );
         while ( elements.hasNextInt() )
         {
            int number = elements.nextInt();
            System.out.println( number );
         }
         elements.close();
      }
      in.close();
   }
}
```

IntegerInput.java Example Output

```
60 70 56 33
60 70 56 33
60
70
56
33
12 23 34
12 23 34
12
23
34
<ENTER>
```

- Arithmetic
- Decision
- Loop Control
- Keyboard Input
- **Command Line Input**
- Characters

Command Line Arguments

- Command line arguments are placed in the **args** parameter of the **main** method.
- Only **Java applications** receive command line arguments.
- The first command line argument is placed in the first element of the **args** array i.e. **args[0]**.
- Use **args.length** to get the number of arguments (the length of the array)

Example using Command Line Arguments

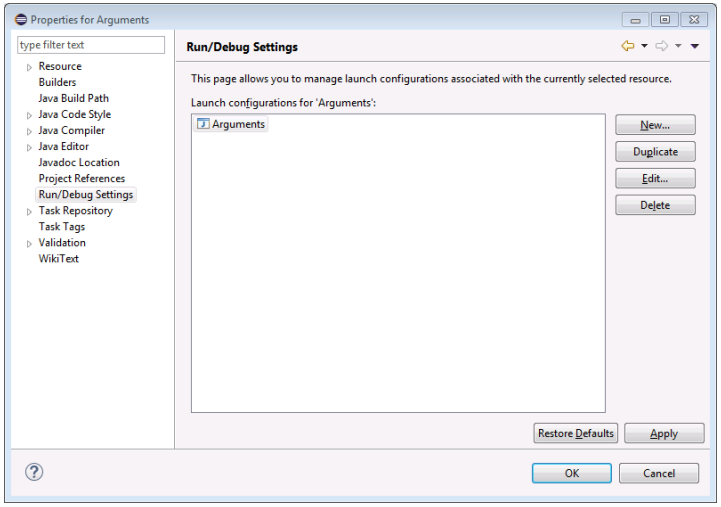
```
// Command Line Arguments
public class Arguments
{
   public static void main( String[] args )
   {
      if ( args.length > 0 )
      {
         for ( int i = 0; i < args.length; i++ )
         {
            System.out.println( "args[ " + i + " ] contains " + args[ i ] );
         }
      }
      else
      {
         System.out.println( "No arguments supplied" );
         System.exit( 1 );
      }
      System.exit( 0 );
   }
}
```

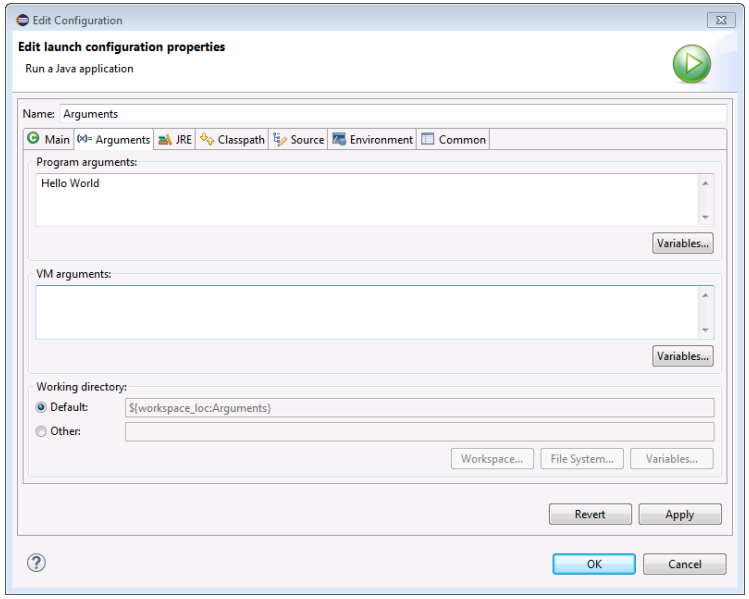
Example Output (java Arguments Hello World)

```
args[ 0 ] contains Hello
args[ 1 ] contains World
```

Command Line Arguments using Eclipse

- To specify command line arguments using Eclipse:
  - Right click on the project in the Package Explorer, and choose **Properties**
  - Then choose **Run/Debug Settings** category, Click **Edit** button.
  - In the Edit Configuration dialogue box, click **Arguments** tab, and enter **Program arguments**.





Contents of **Compare.java**

```
// Comparison of two strings
public class Compare
{
   public static void main( String[] args )
   {
      if ( args.length == 2 )
      {
         if ( args[ 0 ].compareTo( args[ 1 ] ) == 0 )
         {
            System.out.println("Arguments are identical" );
         }
         else
         {
            System.out.println("Arguments are different" );
         }
      }
      else
      {
         System.out.println("Two arguments must be supplied" );
         System.exit( 1 );
      }
      System.exit( 0 );
   }
}
```

Compare.java Example Output

- `java Compare Hello Hello`

```
Arguments are identical
```

- `java Compare Hello World`

```
Arguments are different
```

- Arithmetic
- Decision
- Loop Control
- Keyboard Input
- Command Line Input
- **Characters**

The **Character** Class

- **Character** class
  - Most methods are **static** that take at least a character argument and perform either a test or a manipulation of the character.
  - A constructor that receives a **char** argument to initialise a **Character** object.
  - It has several **non static** methods that operate on an object of the class

Converting between **char** and value (**int**)

Contents of **CharacterValue.java**

```
public class CharacterValue
{
   public static void main( String[] args )
   {
      char ch = 'A';
      System.out.println( ch );
      System.out.println( (int) ch );
   }
}
```

Output:

```
A
65
```

Converting between **char** and value (**int**) (cont.)

Contents of **PrintableCharacter.java**

```
public class PrintableCharacter
{
   public static void main( String[] args )
   {
      int number = 65;
      System.out.println( number );
      System.out.println( (char) number );
   }
}
```

Output:

```
65
A
```

Digit to Character

- Contents of **Radix.java**

```
public class Radix
{
   public static void main( String[] args )
   {
      int digit = 11;
      System.out.println( Character.forDigit( digit, 16 ) );
      char ch = 'f';
      System.out.println( Character.digit( ch, 16 ) );
   }
}
```

Output:

```
b
15
```

Search for a Character

- String.charAt(index) obtains a character (**char**) at position index (0-based)

```
public class Search
{
   public static void main( String[] args )
   {
      int numChars, element, position = -1;
      String number = "203.49";
      String whole, decPlaces;
      char ch;
      numChars = number.length();
      for ( element = 0; element < numChars; element++ )
      {
         ch = number.charAt( element );
         if ( ch == '.' )
            position = element;
      }
      whole = number.substring( 0, position );
      decPlaces = number.substring( position + 1 );
      System.out.println( whole );
      System.out.println( decPlaces );
   }
}
```

Search.java Output

```
203
49
```

Using **static** methods of the **Character** class

- Static methods of **Character** class
  - forDigit(int digit, int radix): the character representation of digit in the given radix
  - digit(char ch, int radix): the numeric value of the character in the given radix
  - isDigit(char ch): determines if the character is a digit
  - isLetter(char ch): determines if the character is a letter
  - isLowerCase, isUpperCase etc.
  - toLowerCase, toUpperCase: converts the character to lower/upper case

Using **static** methods of the **Character** class (cont.)

- Contents of **Vowels.java**

```
public class Vowels
{
   public static int vowelCount( String phrase )
   {
      int numberOfVowels = 0;
      int length = phrase.length();
      char ch;
      for ( int i = 0; i < length; i++ )
      {
         ch = Character.toUpperCase( phrase.charAt( i ) );
         if ( ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U' )
            numberOfVowels++;
      }
      return numberOfVowels;
   }
   public static void main( String[] args )
   {
      String word = "Elephant";
      System.out.print( "Number of vowels in '" + word + "' is " );
      System.out.println( vowelCount( word ) );
   }
}
```

Output:

Number of vowels in 'Elephant' is 3

Using **static** methods of the **Character** class (cont.)

- Contents of **Validate.java**

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
public class Validate
{
   public static boolean checkInt( String input )
   {
      boolean valid = true;
      int length = input.length();
      char ch;
      for ( int i = 0; i < length; i++ )
      {
         ch = input.charAt( i );
         if ( ! Character.isDigit( ch ) )
            valid = false;
      }
      return valid;
   }
```

Using **static** methods of the **Character** class (cont.)

```
 public static void main( String[] args )
   {
      BufferedReader reader = new BufferedReader( new InputStreamReader ( System.in ) );
      String inputLine = "";
      try
      {
         System.out.println( "Please enter a number" );
         inputLine = reader.readLine();
      }
      catch ( IOException ioe )
      {
         System.out.println( ioe );
         System.exit( 1 );
      }
      if ( ! checkInt( inputLine ) )
      {
         System.out.println( inputLine + " is not a valid non-negative integer" );
         System.exit( 2 );
      }
      else
      {
         System.out.println( inputLine + " is a valid non-negative integer" );
         System.exit( 0 );
      }
   }
}
```

Validate.java Example Output

- Example 1:

```
Please enter a number
1234567890
1234567890 is a valid non-negative integer
```

- Example 2:

```
Please enter a number
-30
-30 is not a valid non-negative integer
```

Use of **Escape** Characters

- \ in source code for a String is used to 'escape' the next character to mean some special character
  - \t means a tab character
  - \n means newline
  - to put a single \ in a string as itself, you need \\
  - Arbitrary Unicode characters in a string using \udddd, e.g. \u03c0 is the π character
- When the terminal window is used, to make sure you see Unicode correctly in printed output, you need to specify UTF8 encoding when running the program (e.g. by setting the `JAVA_TOOL_OPTIONS` environment variable to `-Dfile.encoding=UTF8`) or use `java -Dfile.encoding=UTF8`

Example: **Escapes.java**

```
//
// Printing of a tab and a backslash
//
public class Escapes
{
   public static void main( String[] args )
   {
      System.out.print( "Hello\t" );
      System.out.print( "World\n" );
      System.out.println( "The path is C:\\WINNT" );
      System.out.println( "\u03c0 is close to 3.142" );
   }
}
```

Output:

```
Hello   World
The path is C:\WINNT
π is close to 3.142
```

Example: **SplitAndJoin.java**

```
// Reverse the contents of a string
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;

public class SplitAndJoin
{
   public static void main( String[] args )
   {
      BufferedReader reader = new BufferedReader(
                     new InputStreamReader( System.in ) );
      String inputString = "";
      String reversedString = "";
      try
      {
         System.out.println( "Please enter a string" );
         inputString = reader.readLine();
      }
```

Example: **SplitAndJoin.java** (cont.)

```
      catch ( IOException ioe )
      {
         System.out.println( ioe );
         System.exit( 1 );
      }
      for ( int i = inputString.length() - 1; i >= 0; i-- )
      {
         char ch = inputString.charAt( i );
         reversedString += String.valueOf( ch );
      }
      System.out.println( "Reversed string is" );
      System.out.println( reversedString );
      System.exit( 0 );
   }
}
```

Example Output:

```
Please enter a string
Hello
Reversed string is
olleH
```

Note: There is a predefined reverse() method (we will discuss this later).

Example: **GetValidNumber.java**

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;

public class GetValidNumber
{
   //
   // check keyboard input is a valid non-negative integer
   //
   public static boolean checkInt( String input )
   {
      boolean valid = true;
      int length = input.length();
      for ( int i = 0; i < length; i++ )
      {
         char ch = input.charAt( i );
         if ( ch < '0' || ch > '9' )
            valid = false;
      }
      return valid;
   }
```

Example: **GetValidNumber.java** (cont.)

```
public static void main( String[] args )
  {
     BufferedReader reader;
     reader = new BufferedReader( new InputStreamReader( System.in ) );
     boolean valid = true;
     String inputLine = "";
     do
     {
        System.out.println( "Please enter a number" );
        try
        {
           inputLine = reader.readLine();
        }
        catch ( IOException ioe )
        {
           System.out.println( ioe );
           System.exit( 1 );
        }
        valid = checkInt( inputLine );
     } while ( ! valid );
     System.out.println( "Number entered was " + inputLine );
     System.exit( 0 );
  }
}
```

GetValidNumber.java Example Output

```
Please enter a number
abc
Please enter a number
30
Number entered was 30
```