# SWEN20003
# Object Oriented Software Development

## A Quick Tour of Java

**Bach Le**
bach.le@unimelb.edu.au

University of Melbourne

## The Road So Far

**Lectures**

- Subject Introduction

## Learning Outcomes

Upon completion of this topic you will be able to:

- Identify some of the key Java features
- Understand the following in context of Java:
  - Identifiers, Data Types, Variables and Constants
  - Operators and Expressions
  - Flow of control
- Write simple Java programs

## A Brief History of Java

- A programming language developed by Sun Microsystems
- The project started in 1991 by a team led by James Gosling

James Gosling

- The goal of the project at the time was to develop a language that was suitable for the "next wave" in computing – which was expected to be the digitally controlled consumer devices (embedded systems)
- The language was then named 'Oak'

## A Brief History of Java cont..

- In 1993, there was a demise in digital consumer devices and the direction was re-focussed
- The team focussed on the “next-wave” - the Internet
- They decided to work on an embedded language for the web browser – named it an *applet* (a small application)
- *JavaScript, although the name has similarities has nothing to do with Java*

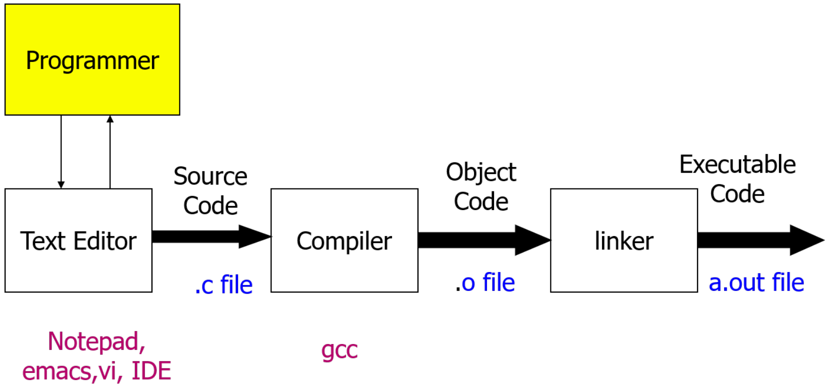
## Java Features

Following are some the key features of Java which we will introduce in this topic:

- Compiled and Interpreted
- Platform-Independent and Portable
- Object Oriented

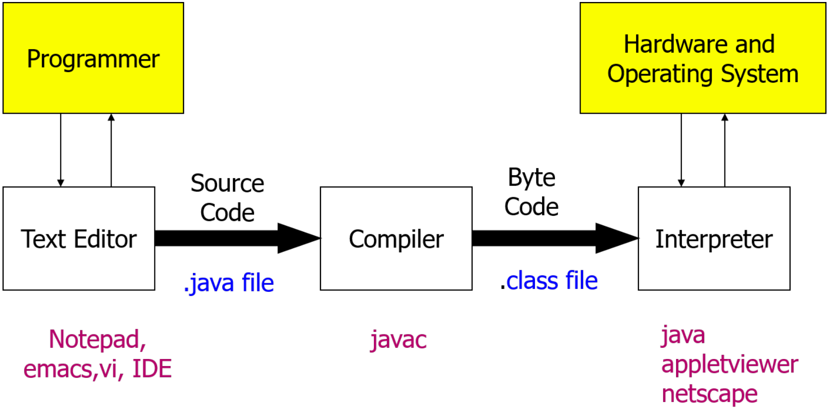
We will introduce more features as we learn new topics during the lectures.

## Compiled Languages (example C)



IDE – Integrated Development Environment

## Java is Compiled and Interpreted



## Java is Compiled and Interpreted

**Java Compiler** converts java *source code* (file with extension .java) to *bytecode* (file with extension .class).

Bytecode is an intermediate form, closer to machine representation.

**An Interpreter** (virtual machine) on any target platform **interprets the bytecode.**

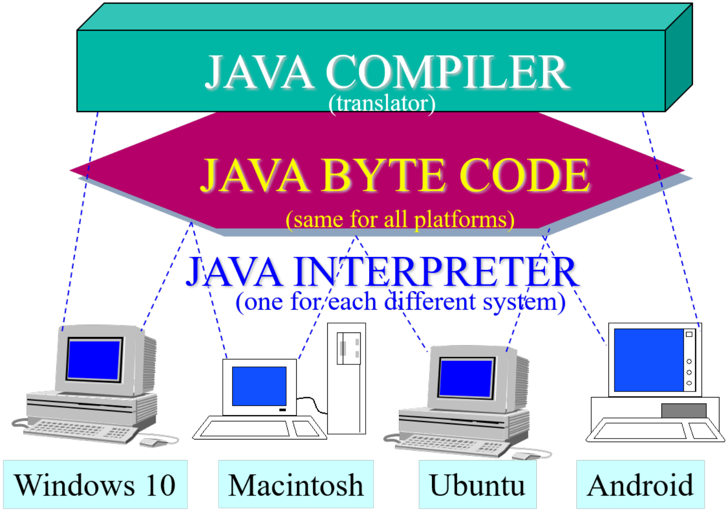
Porting a java system to any new platform involves writing an interpreter.

The interpreter will figure out the equivalent machine dependent code to run.

## Java Features

- Compiled and Interpreted
- Platform-Independent and Portable
- Object Oriented

## Platform Independent



## Java Features

- Compiled and Interpreted
- Platform-Independent and Portable
- Object Oriented

## Object Oriented

Java is an *Object Oriented Programming (OOP)* language.

Common programming constructs are: Classes, Objects, Methods etc.

We will learn more in the coming lectures!

## Applications vs Applets

There are two types of Java programs:

- Application
  - Is a stand-alone program.
  - Has a “main” method.
  - Can be invoked from command line using the Java interpreter.
- Applet
  - A program embedded in a web page.
  - Has no main method.
  - Can be be run by a Java enabled web browser.

In the context of this subject we will only study Java Applications; Java Applets is a bit of an outdated technology - hardly used today.

## Building your first Java Program

Let us try it out!

## Hello World in Java

```java
// HelloWorld.java: Display "Hello World!" on the screen

import java.lang.*;

public class HelloWorld {

    public static void main(String args[]) {

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

    }

}
```

## Hello World in Java

```java
// HelloWorld.java: Display "Hello World!" on the screen

import java.lang.*;

public class HelloWorld {

    public static void main(String args[]) {

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

    }

}
```

Program Output:

```
Hello World!
```

## Hello World in Java and compared with C

```java
// HelloWorld.java: Display "Hello World!" on the screen
import java.lang.*;
public class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello World!");
        return;
    }
}
```

```c
/* helloworld.c: Display "Hello World!" on the screen */
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Hello World! \n");
    return 0;
}
```

## Hello World - Line by Line

**Line 1:**

- This is a comment in Java, similar to line 1 of the C program
- Java supports 3 types of comments:
  - /* */ - Usually used from multi-line comments, similar to C.
  - // - Used for single line comments.
  - /** */ - Documentation comments, will learn more later.

**Line 2:** `import java.lang.*;`

- Serves the same purpose as the #`include` statement used in C.
- Is used to import additional *classes* (similar to libraries used in C).
- In Java, classes are grouped into *packages*.
- Packages may be defined by different people and may even have same class and method names, but they differ by package name (will learn more later): e.g. `ibm.mathlib.*`, `microsoft.mathlib.*`
- By default Java imports `java.lang.*` package, therefore, this statement is optional.

## Hello World - Line by Line

**Line 3:** `public class HelloWorld {`

- Class definition - in Java everything is defined in a class (we will learn more about classes, and the keyword `public` later).
- The name of the class must be the same as the Java file name (`HelloWorld` class must be saved in a file `HelloWorld.java`)

**Line 4:** `public static void main(String args[])`

- Definition of the `main` method, very similar to C.
- A standalone Java program must have a `main` method.
- A class can have only one `main()` method.
- We will learn about keywords `public` and `static` later.
- `String args[]`: Defines command line arguments, similar to C.

## Hello World - Line by Line

**Line 5:** `System.out.println("Hello World!");`

- Serves the same purpose as the `printf` function in C, except `println` is called a *method*, as opposed to a *function* in C.
- `System.out`: `out` is an *object* in the *class* `System`; this class is defined in the `java.lang` package - you will learn more about classes, objects, methods and packages in the coming lectures.

**Line 6:** `return`

- This is optional, and usually not included, just included here for comparison with C.

## Compiling and Running HelloWorld.java

1. Write the Java program using a text editor (e.g. notepad, vim), and save in a file `HellowWorld.java` - you will learn to use an Integrated Development Program (IDE) later
2. Ensure that the java build and runtime environment is installed on the machine.
   - Open a command window and type the commands:

     `javac -version, java -version`
3. Compile the program using the following command:

   ```
   javac HelloWorld.java
   ```

   The command if successful will generate a file `HelloWorld.class`
4. Run the program using the following command:

   ```
   java HelloWorld
   ```

   You should see the following output:

```
Hello World!
```

## Command Line Arguments

If you run `java HelloWorld` with `command line arguments as follows:`

```
java HelloWorld Asutralia England France
```

`args[] (defined in public static void main(String args[]))` will contain the command line argements:

```
args[0] -> Australia
args[1] -> England
args[2] -> France
```

## Command Line Arguments - Example

```java
// CommandLineTest.java - Program with command line arguments
public class CommandLineTest {
    public static void main(String args[]) {
        int count, i=0;
        count = args.length;
        System.out.println("Number of arguments = " +  count);
        while(i < count) {
            System.out.println("arg[" + i + "]: "   + args[i]);
            i = i + 1;
        }
    }
}
```

If you run: `java CommandLineTest Australia England France`
Program Output:

```
Number of arguments = 3
arg[0]: Australia
arg[1]: England
arg[2]: France
```

## Differences between Java and C

Java is an Object Oriented language : C is a Procedural Language.

Java has no goto, sizeof and typedef statements.

Java has no structures and unions.

Java has no explicit pointer type.

Java has no Preprocessor: no `#define`, `#include`, `#indef`

Java is safe and well defined: e.g. memory is managed by the virtual machine not by the programmer.

## Identifiers, Data Types, Variables and Constants

## Identifiers

**Keyword**

*Identifier:* A name that uniquely identifies a program element such as a class, object, variable, method.

Java identifier **rules**:

- must not start with a digit
- all the characters must be letters, digits, or the underscore symbol
- can theoretically be of any length
- are case-sensitive: `Rate`, `rate`, and `RATE` are different variables

Java identifier **conventions**:

- variables, methods, and objects: start with a lower case letter, indicate "word" boundaries with an uppercase letter, and restrict the remaining characters to digits and lowercase letters (e.g. `topSpeed`, `bankRate`, `timeOfArrival`)
- classes: start with an upper case letter and, otherwise, adhere to the rules above (e.g. `PrintDemo`, `HelloWorld`)

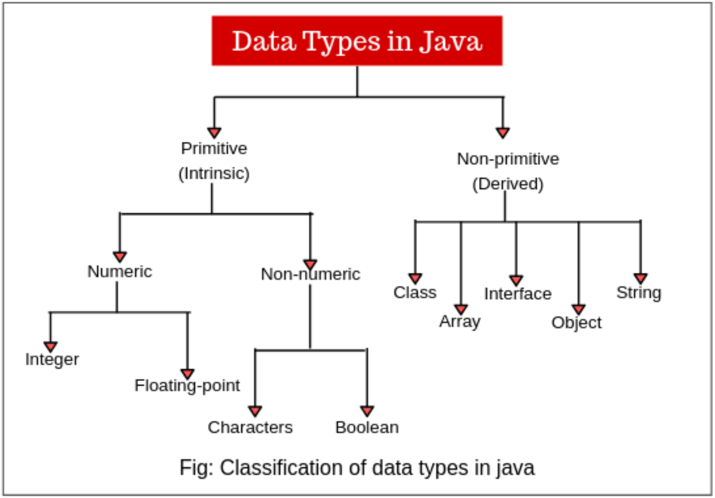
## Identifiers

Identifiers that have a predefined meaning in Java, such as **keywords** and **reserved words**, must not be used as as identifiers in your programs:
e.g. `public, class, void, static`

Identifiers that are defined in libraries required by the Java language standard packages are **predefined identifiers**;

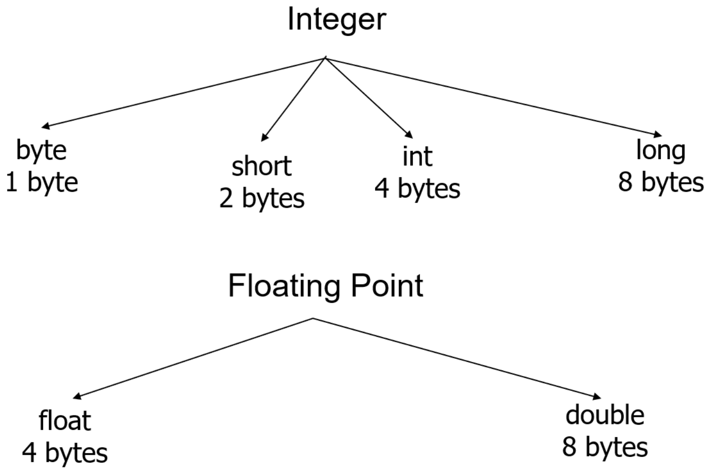
- although they can be redefined, this could be confusing and dangerous because doing so would change their standard meaning.
  e.g `System, String, println`

## Java Data Types



From: `https://www.scientecheasy.com/2018/06/data-types-in-java-primitive-example.html`

## Java Data Types



## Java Data Types

Floating point numbers:

- `float` type has single-precision.
- `double` type has double-precision.
- Generally, floating point numbers are treated as double-precision numbers. To force them to be in single precision we must append f or F to the number: e.g. `float a = 2.3F; double b = 6.7;`

Boolean numbers:

- Java `boolean` type variables can hold a `true` or `false`: e.g. `boolean x = true;`

## Variables

**Keyword**

*Variable:* Refers to information that is stored in program memory that can be changed; a variable has a memory location and an identifier.

- Variables must be *declared* and *initialized* before use.
  Syntax:

```
    <type> <variable name> = <initial value>;
```

Examples:

```
    int   count = 1;
    float length = 2.3F;
    double height = 6.7;
    boolean status = true;
```

## Variables

- *Assignment operator* (=) is used to change the value of a variable.
  Syntax:

```
<variable name> = <other variable name> OR  <value> OR <expression>;
```

  Examples:

```
    int countX = 1, countY = 2;
    countX = countY;
    countX = countY + countX;
    countX += 3; // Shorthand operator countX = countX + 3;
```

- In general, the value of one type cannot be stored in a variable of another type, but there are exceptions:
  Examples:

```
    int intVariable = 2.99; // Not a valid assignment
    double doubleVariable = 2; // Is a valid assignment
```

## Variables

- A value of any type in the following list can be assigned to a variable of any type that appears to the right of it:

```
byte -> short -> int -> long -> float -> double
char -> int
```

- An explicit **type cast** is required to assign a value of one type to a variable whose type appears to the left of it on the above list (e.g., double to int). Examples:

```
int x = 2.99; // Not a valid assignment
int y = (int)2.99; // Is a valid assignment; y will be 2 not 3
```

- Variables of type `int` cannot be assigned to a variable of type `boolean`, nor can a variable of type `boolean` be assigned to a variable of type `int`.

## Variables

- Java variables are classified into three categories.
  - **instance** variables
  - **static** (or **class**) variables
  - **local** variables

Keyword

*Local Variables:* Variables defined inside a Java method.

We will introduce *instance* and *static* variables in the coming lectures.

## Constants

**Keyword**

*Constant:* A value that does not change during the execution of the program; also called “READ” only values.

- Constants are declared with the Java key word `final`.
  Examples:

```
final  int MAX_LENGTH = 420;
final  double  PI = 3.1428;
final  char  CHAR_CONSTANT = 'Z';
final  boolean BOOL_CONSTANT = true;
final  String STRING_CONSTANT = "Welcome to Java";
```

- By convention upper case letters are used for defining constants.
- Data type must be explicitly specified when defining constants; this is not required in C.

## Variables and Constants - Example Program

```java
// ConstantsVariablesExample.java
// Demonstrates the use of constants and variables
public class ConstantsVariablesExample {
    public static void main(String args[]) {
        final double PI = 3.1428;
        // All the variables below are local variables
        float radius = 5.5F;
        double area;
        boolean x = true;
        area = PI * radius * radius;
        System.out.println("Circle Radius = "
                    + radius + " Area = " + area);
        System.out.println("Boolean Value = " + x);
    }
}
```

Program Output:

```
Circle Radius = 5.5 Area = 95.0697
Boolean Value = true
```

## Operators and Expressions

## Operators

Java Operators can be classified into the following related categories.

- Arithmetic
- Relational
- Logical
- Assignment
- Increment and decrement
- Conditional
- Bitwise
- Special

## Arithmetic Operators

| Operator | Meaning |
|---|---|
| + | Addition and unary plus |
| − | Subtraction or unary minus |
| ∗ | Multiplication |
| / | Division |
| % | Modulo division |

**Note:** When one of the operands is real and the other is an integer, the expression is called a mixed-mode arithmetic expression. If either operand is of real type, then the other operand is also converted to real and real arithmetic is performed.

## Arithmetic Operators - Example

```java
// ArithmeticExample.java - Using arithmetic operators
public class ArithmeticExample {
    public static void main(String args[]) {
        float a = 20.5F, b = 6.4F;
        int c = 11, d = 5;
        System.out.println("a + b = " + (a+b));
        System.out.println("a - b = " + (a-b));
        System.out.println("a*b = " + (a*b) );
        System.out.println("a/b = " + (a/b));
        System.out.println("c%d = " + (c%d));
    }
}
```

Program Output:

```
    a + b = 26.9
    a - b = 14.1
    a*b = 131.2
    a/b = 3.203125
    c%d = 1
```

## Relational Operators

| **Operator** | **Meaning** |
|---|---|
| $<$ | Is less than |
| $<=$ | Is less than or equal to |
| $>$ | Is greater than |
| $>=$ | Is greater than or equal to |
| $==$ | Is equal to |
| $! =$ | Is not equal to |

**Note:** The result of a relational operator is type `boolean`.

## Relational Operators - Example

```java
// RelationalExample.java - Using relational operators
public class RelationalExample {
    public static void main(String args[]) {
        int a = 3, b = 5;
        System.out.println("a < b = " + (a<b));
        System.out.println("a > b = " + (a>b));
        System.out.println("a <= b = " + (a<=b));
        System.out.println("a >= b = " + (a>=b));
        System.out.println("a == b = " + (a==b));
        System.out.println("a != b = " + (a!=b));
    }
}
```

Program Output:

```
a < b = true
a > b = false
a <= b = true
a >= b = false
a == b = false
a != b = true
```

## Logical Operators

| Operator | Meaning |
|---|---|
| && | Logical AND |
| \|\| | Logical OR |
| ! | Logical NOT |

| op1 | op2 | op1 && op2 | op1 \|\| op2 | !op1 |
|---|---|---|---|---|
| true | true | true | true | false |
| true | false | false | true | false |
| false | true | false | true | true |
| false | false | false | false | true |

## Bitwise Operators

| **Operator** | **Meaning** |
|---|---|
| & | Bitwise AND |
| \| | Bitwise OR |
| ^ | Bitwise exclusive OR |
| ~ | One's compliment |
| << | Shift Left |
| >> | Shift Right |
| >>> | Shift Right with zero fill |

## Other Operators

Increment and Decrement Operators:

- ++ and −−
  Example:

```
    ++x; x++; --x; x--;
```

Conditional Operators:

- `exp1 ? exp2 : exp3`
  Example:

```
    x = (a > b) ? a : b;
```

This is the same as:

```
    if  (a > b)
        x = a;
    else
        x = b;
```

# Mathematical Functions

Java supports mathematical functions such as `cos`, `sin`, `log` using the Math class, defined in the `java.lang` package.
The functions should be used as follows:

- `Math.method_name();`
  Example:

```
double y, z;
y = Math.sqrt(x);
z = Math.cos(y);
```

**Note:** Refer to Java documentation for a complete list of methods supported.

## Mathematical Functions - Example

```java
// MathSqrtExample.java - Compute the square root of a number
class MathSqrtExample {
    public static void main(String args[]) {
        double x = 4;
        double y;
        y = Math.sqrt(x);
        System.out.println("The square root of " + x + " is " + y);
    }
}
```

Program Output:

```
The square root of 4.0 is 2.0
```

## Flow of Control

## Flow of Control

**Keyword**

*Flow of Control:* Refers to branching and looping mechanisms in the Java language.

Most branching and looping statements are controlled by boolean expressions:

- A boolean expression evaluates to either true or false.
- The primitive type boolean may only take the values true or false

## Branching

Java supports the following branching statements:

- `if-else` statement
- multi-way `if-else` statement
- `switch` statement
- two way decision expression

## The if-else and multi-way if-else Statements

```
if (Boolean_Expression) {
    Yes_Statements
} else {
    No_Statements
}
```

```
if (Boolean_Expression) {
    Yes_Statements_1
} else if (Boolean_Expression) {
    Yes_Statements_2
} else if (Boolean_Expression) {
    Yes_Statements_3
} else {
    Statements_For_All_Other_Possibilities
}
```

## The if-else Statement - Example

```java
//IfElseExample.java - Demonstrates  if-else control flow
public class IfElseExample {
    public static void main(String args[]){
        int i = 5;
        if (i < 10) {
            System.out.println("i is less than 10");
        }
        else {
            System.out.println("i is greater than or equal to 10");
        }
    }
}
```

Program Output:

```
i is less than 10
```

## The switch Statement

The switch statement supports *multi-way branching*:

```
switch (Controlling_Expression)
{
    case Case_Label_1:
        Statement_Sequence_1
        break;
    case Case_Label_2:
        Statement_Sequence_2
        break;
    case Case_Label_n:
        Statement_Sequence_n
        break;
    default:
        Default_Statement Sequence
        break;
}
```

## The switch Statement - Example

```
//SwitchExample.java - Demonstrates the switch statement
public class SwitchExample {
    public static void main (String args[]) {
        char x = 'b';
        switch(x) {
            case 'a':
                System.out.println("Chracter = a");
                break;
            case 'b':
                System.out.println("Character = b");
                break;
            default:
                System.out.println("Other character");
                break;
        }
    }
}
```

Program Output:

```
    Character = b
```

## The Two-way Decision Statement - Example

```
// TwoWayExample.java - Demonstrates two way decision making
class TwoWayExample {
    public static void main(String args[]) {
        int i = 5;
        int flag;
        flag = (i < 10) ? 0 : 1;
        System.out.println("Flag = " + flag);
    }
}
```

Program Output:

```
Flag = 0
```

## Loops

Loops in Java are similar to those in other high-level languages.

- The code that is repeated in a loop is called the body of the loop.
- Each repetition of the loop body is called an iteration of the loop.
- Loops can be nested similar to other programming languages.

Java supports the following looping statements.

- `while` statement
- `do-while` statement
- `for` statement

## The while and do-while Statements

```
while (Boolean_Expression) {
    Statement_1
    Statement_2
    ....
    Statement_Last
}
```

```
do {
    Statement_1
    Statement_2
    ...
    Statement_Last
}  while (Boolean_Expression);
```

## The while Statement - Example

```java
// WhileExample.java - Demonstrates the use of the while statement
public class WhileExample {
    public static void main(String args[]) {
        int sum = 0, n = 1;
        while (n <= 10) {
            sum = sum + n;
            n = n + 1;
        }
        System.out.println("Sum = " + sum);
    }
}
```

Program Output:

```
Sum = 55
```

## The do-while Statement - Example

```
// DoWhileExample.java - Demonstrates the use of the do-while statement
public class DoWhileExample {
    public static void main(String[] args) {
        int sum = 0, n = 1;
        do {
            sum = sum + n;
            n = n +1;
        } while ( n <= 10);
        System.out.println("Sum = " + sum);
    }
}
```

Program Output:

```
    Sum = 55
```

## The for Statement

```
for (Initialize_Expressions; Terminate_Expression; Update_Expressions) {
    Statement_1
    Statement_2
    ....
    Statement_Last
}
```

Contains three types of expressions within the parentheses related to controlling variables:

- `Initialize_Expressions` - determine how the control variable or variables are *initialized* or *declared and initialized* before the first iteration
- `Terminate_Expression` - determines when the loop should end, based on the evaluation of a Boolean expression *before* each iteration
- `Update_Expressions` - determine how the control variable or variables are *updated* after each iteration of the loop body

## The for Statement - Example

```java
// ForExample.java - Demonstrates the for loop
public class ForExample {
    public static void main(String[] args) {
        for (int n = 0; n < 5; n++) {
            System.out.println(" n = " + n);
        }
    }
}
```

Program Output:

```
n = 0
n = 1
n = 2
n = 3
n = 4
```

## The break Statement

The break statement causes to exit the loop (while, do or for).

```
1       while(.....) {
2           ........
3           if (condition)
4               break;   // Will go to line 7
5           ........
6       }
7       ....
```

```
1       loop1: while(.....) {
2           .......
3           loop2:  while (.....) {
4               ........
5               if (condition)
6                   break loop1; // Will go to line 11
7               ........
8           }
9           ..........
10      }
11      .............
```

## The break Statement - Example

```
// BreakExample.java - Demonstrates the use of the break statement
public class BreakExample {
    public static void main(String[] args) {
        loop1: for (int i = 0; i < 3; i++) {
            loop2: for (int j = 0; j < 3; j++) {
                System.out.println("i=" + i + " j=" +j);
                if (j == 1)
                    break loop1;
            }
        }
    }
}
```

Program Output:

```
i=0 j=0
i=0 j=1
```

## The continue Statement

The `continue` statement skips the rest of the statements in the loop (`while`, `do` or `for`).

```
1       while (.....) {
2           ........
3           ........
4           if (condition)
5               continue; // Will goto line 1
6           ........
7           ........
8       }
9       .............
```

## The continue Statement - Example

```java
// ContinueExample.java - Demonstrates the use of the continue statement
public class ContinueExample {
    public static void main(String[] args) {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (j == 1)
                    continue;
                System.out.println("i=" + i + " j=" +j);
            }
        }
    }
}
```

Program Output:

```
    i=0 j=0
    i=0 j=2
    i=1 j=0
    i=1 j=2
    i=2 j=0
    i=2 j=2
```

## Learning Outcomes

Upon completion of this topic you will be able to:

- Identify some of the key Java features
- Understand the following in context of Java:
  - Identifiers, Data Types, Variables and Constants
  - Operators and Expressions
  - Flow of control
- Write simple Java programs

## References

- Absolute Java by Water Savitch (Fourth Edition), Chapters 1 & 3