CSIT121

Object Oriented Design and Programming

Lesson 1

Introduction and Review

Textbook

- **Recommended Textbook**
  - Java how to Program (Early Objects), Eleventh Edition,
  - Paul Deitel and Harvey Deitel, Pearson, 2018.

- **Important Reference Books**
  - Introduction to Java Programming Comprehensive, Eleventh Edition,
  - Liang, Pearson 2018

  - Java programming, Ninth Edition, Joyce Farrell,
  - Cengage learning 2018

  - Java programming: From problem solving to design, Fifth edition,
  - D.S Malik, Cengage learning 2012

A simple Java Program

Comments

Each program is defined by a class.

Execution of program starts with the Main() method.

HelloWorld.java

```
1  //This is HelloWorld.java
2
3  public class HelloWorld {
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7          System.out.println("Hello World");
8
9      }
10 }
```

Take not of matching braces {}

Primitive Data Types

- Java's primitive data types are used to represent "simple" values.
- These types are:
  - int, byte, short and long for integer values (whole numbers and their negatives)
  - float and double for real values (fractional numbers)
  - char for characters (letters, digits, symbols and punctuation)
  - boolean for logical values (true and false)

Primitive Data Types

Data Type	Size in bits	Range of Values
boolean	8	true or false
char	16	0 to 65535
byte	8	-128 to +127 (-2^7 to $2^7 - 1$)
short	16	-32,768 to +32,767 (-2^{15} to $2^{15} - 1$)
int	32	-2,147,483,648 to +2,147,483,647 (-2^{31} to $2^{31} - 1$)
long	64	-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 (-2^{63} to $2^{63} - 1$)
float	32	-3.4e38 to +3.4e38 (6 to 7 significant digits)
double	64	-1.7e308 to +1.7e308 (14 to 15 significant digits)

String

- A String variable contains a collection of characters surrounded by double quotes:

```
1  public class StringExampple {
2
3      public static void main(String[] args) {
4          // TODO Auto-generated method stub
5          String greeting = "Hello";
6          String name = "Peter";
7
8          System.out.println(greeting+" "+name);
9      }
10 }
```

- Note: String is a “Non-Primitive” data type.

Variable

- To create a variable, you must specify the type and assign it a value.
- Syntax
  - *type variableName = value;*

```java
public class VariablesExample {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        String name = "Peter";
        int age = 23;
        double weight = 67.5;
        double height = 1.75;
        char gender = 'm';
        boolean isStudent = true;

        System.out.println("Name:"+name);
        System.out.println("Age:"+age);
        System.out.println("Height:"+height+"m");
        System.out.println("Weight:"+weight+"kg");
        System.out.println("Gender:"+name);
        System.out.println("Is Student:"+isStudent);
    }

}
```

Constant

- To prevent overwriting existing values, use Constant value
- Use the final keyword
  - Means unchangeable and read-only
- Naming Convention
  - Use only UPPERCASE letters and separate each word with an underscore “_”.
- Usually is declared as a static class variable

```java
public class ConstantExample {

    final static double CONVERT_RATE = 2.4;

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        double amount = 10;
        double convertedAmount = amount * CONVERT_RATE;
        System.out.println(convertedAmount);
    }
}
```

Enumerations

- Basic enum type defines a set of constants that are represented as unique identifiers.

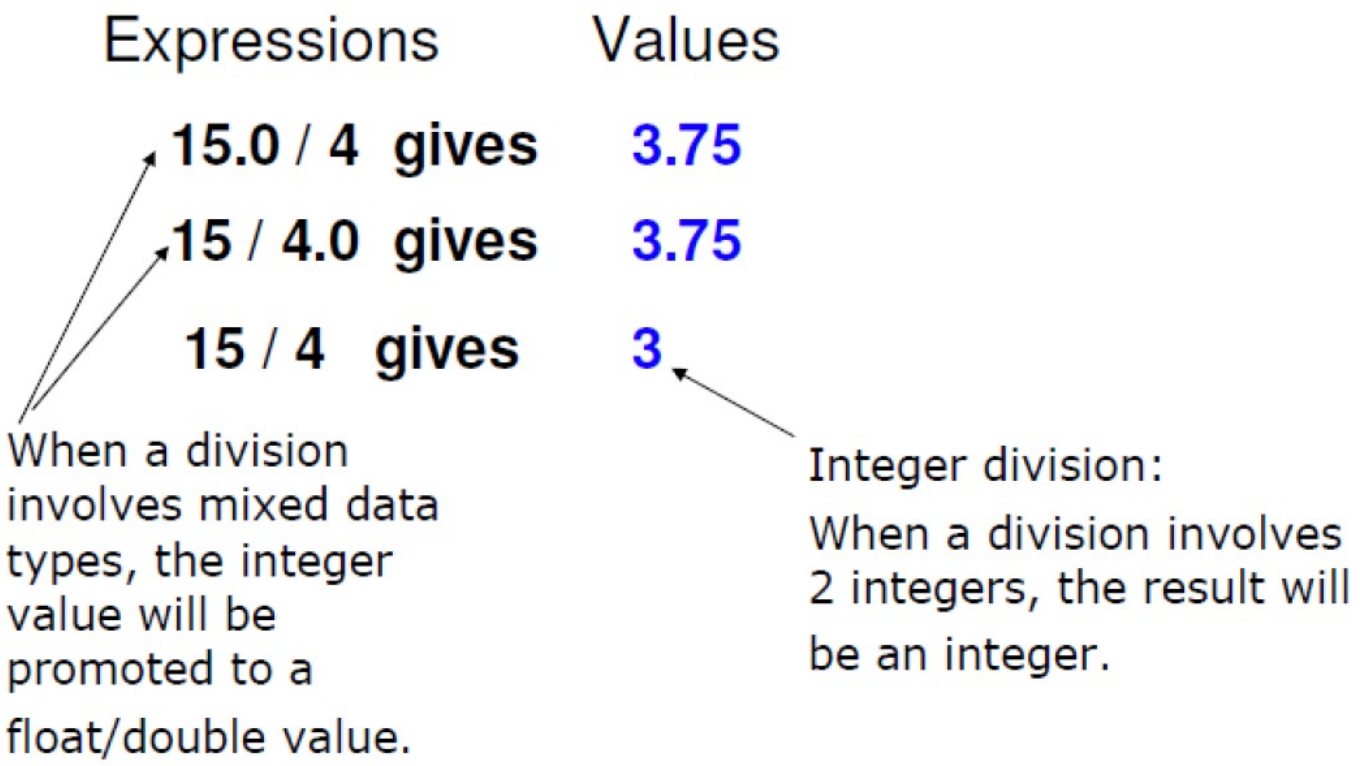
```
1  public class EnumExample {
2
3      enum Status {CONTINUE, WIN, LOST};
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7          Status myStatus = Status.WIN;
8          System.out.println(myStatus);
9      }
10 }
```

Arithmetic operation

Operator	Meaning	Example	Value of z
+	Addition	int z = 17 + 4;	21
-	Subtraction	int z = 17 – 4;	13
*	Multiplication	int z = 17 * 4;	68
/	Division	int z = 17 / 4;	4 *
%	Modulus	int z = 17 % 4;	1
-	Unary minus	int z = -17;	-17

*Division operator for integers returns integer result.

Division of double and int



Precedence of Operator

- All expressions are evaluated according to an operator precedence.
- Any arithmetic operators at the same level of precedence are evaluated from left to right but precedence can be forced by using parentheses.

Highest precedence

Lowest precedence

Operator	Meaning
( )	Parentheses
+, -	Unary operators
* , / , %	Multiplication, division, modulus
+, -	Addition, subtraction
=	Assignment

Increment / Decrement Operators

- Increment operator (++) adds 1 to an integer value.
- Decrement operator (--) subtracts 1 from an integer value.

count++; (post-increment)
++count; (pre-increment)
They are the same as: count = count + 1;

count--; (post-decrement)
--count;(pre-decrement)
They are the same as: count = count -1;

Increment / Decrement Operators

- When increment or decrement operator is used in a larger expression, it can yield different results depending on the form used (prefix or postfix)

```
1  public class IncDecOperator {
2
3      public static void main(String[] args) {
4          // TODO Auto-generated method stub
5          int count = 15;
6          int total = 0;
7
8          total = count++; //post-increment
9          System.out.println(total);//15
10         System.out.println(count);//16
11
12         count = 15;
13         total = 0;
14         total = ++count; //pre-increment
15         System.out.println(total);//16
16         System.out.println(count);//16
17     }
18
19 }
```

Assignment Operators

- Java provides several assignment operators that combines a basic operation with assignment.

`total += 5;`	`total = total + 5;`
`total += (sum - 2)/count;`	`total = total + ((sum-2)/count);`
`result *= num1 + num2;`	`result = result * (num1 + num2);`
`result %= (num - 30)/2;`	`result = result % ((num - 30)/2);`

Datatype Conversion

- Java is a strongly typed language, each data is associated with a particular type.
- Sometimes there is a need to convert a data value of one type to another.
- Conversion can occur in 2 ways:
  - Assignment conversion
  - Casting

Data Conversion - Assignment conversion

- Occurs when a value of one type is assigned to a variable of another type
- For example, int(32 bits) to double(64 bits)

```
1  public class AssignmentConversion {
2
3      public static void main(String[] args) {
4          // TODO Auto-generated method stub
5          int value1 = 32;
6          double value2 = value1;
7          System.out.println(value1);
8          System.out.println(value2);
9      }
10 }
```

```
32
32.0
```

Data Conversion - Casting

- A cast is a Java operator that is specified by a type name in parentheses. It is placed in front of the value to be converted.
- For example, double(64 bits) to int(32 bits) – decimal places removed

```
1  public class CastingConversion {
2
3      public static void main(String[] args) {
4          // TODO Auto-generated method stub
5          double value1 = 32.5;
6          int value2 = (int)value1;
7          System.out.println(value1);
8          System.out.println(value2);
9      }
10 }
```

```
32.5
32
```

Converting String to Numeric Type

- Convert to integer
  - Use Integer.parseInt()
- Convert to double
  - Use Double.parseDouble()
- Note: Integer and Double are Java API classes.

```
1  public class StringToNumericConversion {
2
3      public static void main(String[] args) {
4          // TODO Auto-generated method stub
5          String s1 = "32";
6          int value1 = Integer.parseInt(s1);
7          System.out.println(value1);
8
9          String s2 = "42.8";
10         double value2 = Double.parseDouble(s2);
11         System.out.println(value2);
12     }
13 }
```

User input - Scanner

- Use Scanner to get user input from keyboard
  - nextLine() – for string

```java
import java.util.*;
public class ScannerExample1 {

    public static void main(String[] args) {
        System.out.print("Enter some text:");
        Scanner kb = new Scanner(System.in);
        String text = kb.nextLine();
        System.out.println(text);
    }
}
```

```
<terminated> ScannerExample1 [Java Application] /Libra
Enter some text:apple
apple
```

User input - Scanner

- Use Scanner to get user input from keyboard
  - nextInt() – for int

```java
import java.util.*;
public class ScannerExample2 {

    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Enter int 1:");
        int n1 = kb.nextInt();
        System.out.print("Enter int 2:");
        int n2 = kb.nextInt();
        int total = n1 + n2;
        System.out.println("Total:"+total);
    }
}
```

```
<terminated> ScannerExample2 [Java
Enter int 1:5
Enter int 2:9
Total:14
```

User input - Scanner

- Use Scanner to get user input from keyboard
  - nextDouble – for double

```java
import java.util.Scanner;

public class ScannerExample3 {

    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Enter double 1:");
        double d1 = kb.nextDouble();
        System.out.print("Enter double 2:");
        double  d2 = kb.nextDouble();
        double total = d1 + d2;
        System.out.println("Total:"+total);
    }
}
```

```
<terminated> ScannerExample3 [Java Application]
Enter double 1:12.8
Enter double 2:15.7
Total:28.5
```

User input – Scanner Problem

- Problem -> read int and then String
- After we enter an int and hit enter key “\n”
- nextInt() will not read in “\n”. Left over in the buffer
- The next nextLine() will read in \n and assign to string s.

```java
import java.util.Scanner;

public class ScannerProblem1 {

    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Enter int:");
        int n = kb.nextInt();//will not read in \n
        System.out.print("Enter text:");
        String s = kb.nextLine();//read in the left over\n

        System.out.println("int:"+n);
        System.out.println("String:"+s);
    }
}
```

```
<terminated> ScannerProblem1 [Java Applic
Enter int:8
Enter text:int:8
String:
```

User input – Scanner Problem

- Solution 1 -> read in the left over “\n” before nextLine()

```java
import java.util.Scanner;

public class ScannerProblem2 {

    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.print("Enter int:");
        int n = kb.nextInt();//will not read in \n
        String s = kb.nextLine();//read in the left over\n
        System.out.print("Enter text:");
        s = kb.nextLine(); //read again

        System.out.println("int:"+n);
        System.out.println("String:"+s);
    }
}
```

```
<terminated> ScannerProblem3 [Java Applica
Enter int:5
Enter text:apple
int:5
String:apple
```

User input – Scanner Problem

- Solution 2 -> Always new a Scanner before getting input()
  - Put it in a function
  - Reusable

```
<terminated> ScannerProblem3 [Java Applica
Enter int:5
Enter text:apple
int:5
String:apple
```

```java
import java.util.Scanner;

public class ScannerProblem3 {

    public static void main(String[] args) {
        int n = getInt("Enter int:");
        String s = getString("Enter text:");
        System.out.println("int:"+n);
        System.out.println("String:"+s);
    }

    public static String getString(String msg) {
        Scanner kb = new Scanner(System.in);
        System.out.print(msg);
        return kb.nextLine();
    }

    public static int getInt(String msg) {
        Scanner kb = new Scanner(System.in);
        System.out.print(msg);
        return kb.nextInt();
    }
}
```

Condition

- A condition make up of three element:
  - A value/variable
  - Relational operator
  - A value/variable
- A condition always evaluate to a Boolean (true/false)

```java
public class Condition1 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int a = 5;

        boolean c1 = a >= 3; //5>=3 => true
        boolean c2 = a < 5;  //5<5  => false
        boolean c3 = a == 5; //5==5 => true
        boolean c4 = a == 3; //5==3 => false

        System.out.println(c1);
        System.out.println(c2);
        System.out.println(c3);
        System.out.println(c4);
    }
```

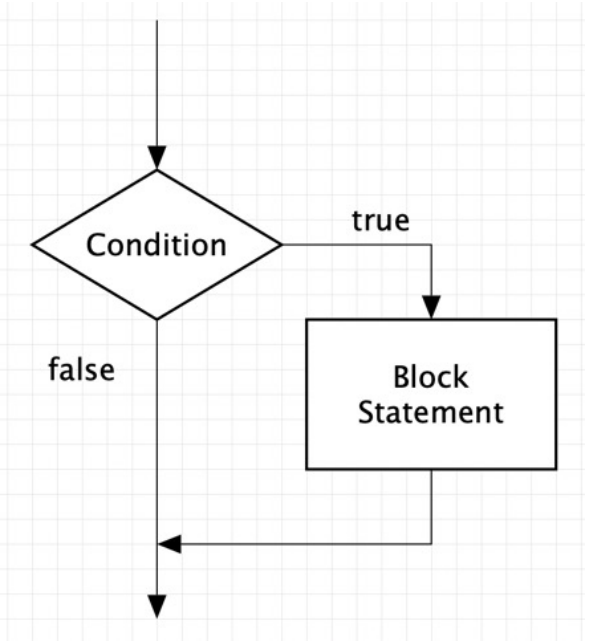
Condition – relational operator

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

Decision - if

- Use condition in an if structure to instruct Java how to make decision.
- If a condition is true, we enter the if block statement.
- Else we skip the if block statement.

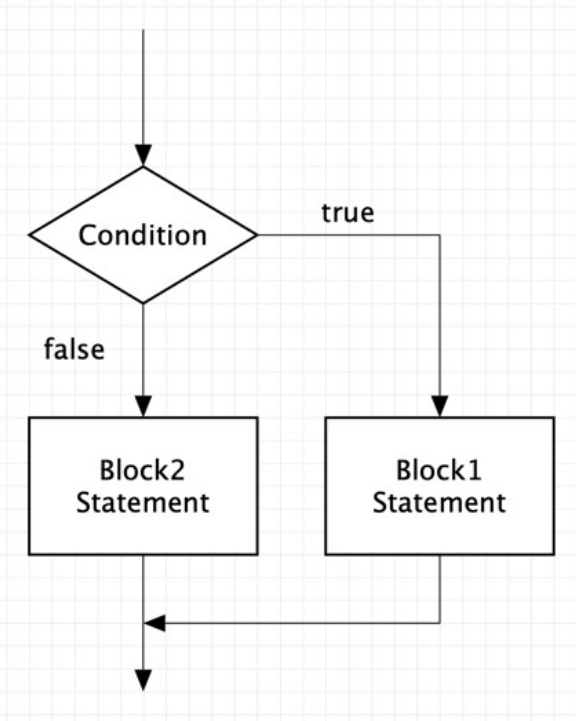
```
1 public class IfExample {
2     public static void main(String[] args) {
3         int n = 6;
4 
5         if(n>5) {
6             System.out.println("n is more than 5");
7         }
8     }
9 }
```



Decision – if else

- If a condition is true, we enter block 1 statement.
- Else we enter block 2 statement

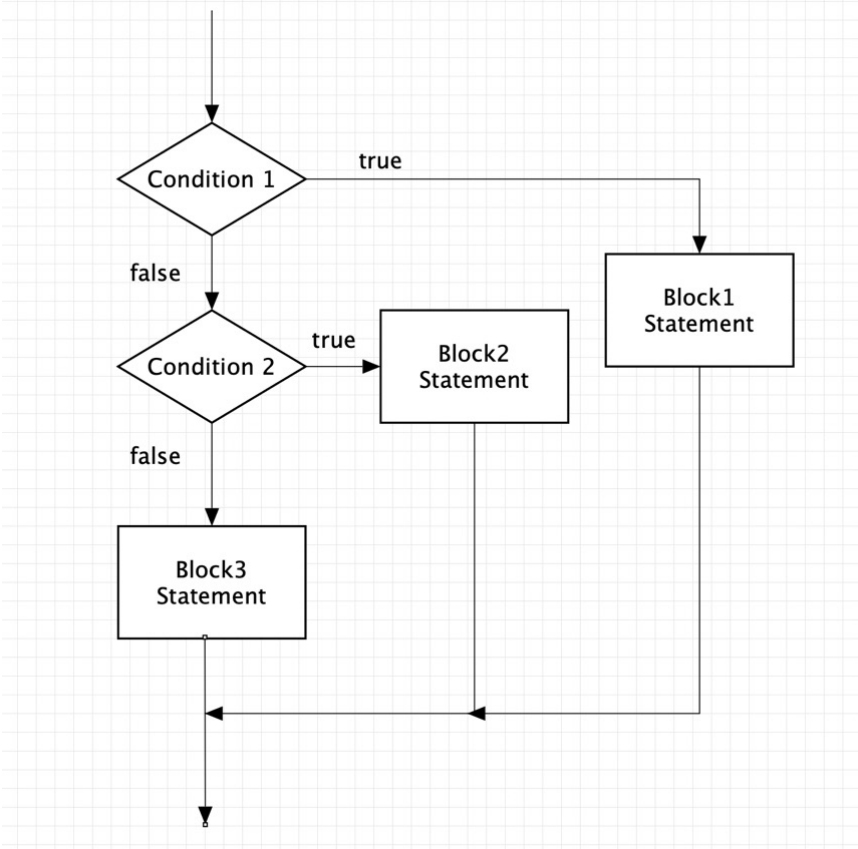
```
1  public class IfElseExample {
2
3      public static void main(String[] args) {
4
5          int n = 3;
6
7          if(n>5) {
8              System.out.println("n is more than 5");
9          }else {
10             System.out.println("n is not more than 5");
11         }
12     }
13 }
```



Decision – if else if else…

- If a condition 1 is true, we enter block 1 Statement,
- Else we check if condition 2 is true, we enter block 2 statement.
- Else we enter block 3 statement.

```
1 public class IfElseExample {
2
3•    public static void main(String[] args) {
4
5         int n = 0;
6
7         if(n>5) {
8             System.out.println("n is more than 5");
9         }else if (n>3){
10            System.out.println("n is more than 3");
11        }else {
12            System.out.println("n is not more than 3");
13        }
14    }
15 }
```



Compound Condition

- Two conditions can be compounded using AND or OR to form a more complex condition.

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

Condition A	Condition B	A || B
false	false	false
false	true	true
true	false	true
true	true	true

Compound Condition

- AND

```java
public class AndExcample {

    public static void main(String[] args) {

        double pressure = 50;

        if(pressure>=40 && pressure<=60) {
            System.out.println("Pressure within range");
        }else {
            System.out.println("Pressure not within range");
        }

    }
}
```

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

Compound Condition

- OR.

```java
public class OrExample {

    public static void main(String[] args) {

        double purchaseAmount = 150;
        boolean isMember = true;

        if(purchaseAmount>=100 || isMember) {
            System.out.println("You got a 10% discount!");
            purchaseAmount *= 0.9;
        }

        System.out.println("Please pay:$"+purchaseAmount);
    }
}
```

Condition A	Condition B	A || B
false	false	false
false	true	true
true	false	true
true	true	true

Decision – switch

- The Java switch
- Fall through:

```
1  public class SwitchExample1 {
2
3      public static void main(String[] args) {
4          // TODO Auto-generated method stub
5          int caseNum = 1;
6
7          switch(caseNum) {
8              case 1 : System.out.println("1");
9              case 2 : System.out.println("2");
10             case 3 : System.out.println("3");
11         }
12     }
13 }
```

Output:

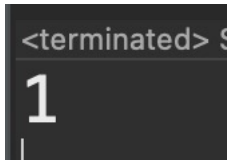
<terminated
123

Decision – switch

- The Java switch
- Need to use “break” to isolate each case.

```
1
2 public class SwitchExample2 {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6         int caseNum = 1;
7
8         switch (caseNum) {
9         case 1:
10            System.out.println("1");
11            break;
12        case 2:
13            System.out.println("2");
14            break;
15        case 3:
16            System.out.println("3");
17            break;
18        }
19    }
20 }
```

Output:



Decision – switch

- The Java switch
- Use default to handle non defined cases.

```java
public class SwitchExample3 {

    public static void main(String[] args) {
        int caseNum = 4;

        switch (caseNum) {
        case 1:
            System.out.println("1");
            break;
        case 2:
            System.out.println("2");
            break;
        case 3:
            System.out.println("3");
            break;
        default:
            System.out.println("None of the above");
        }
    }
}
```

Output:

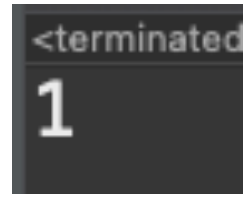
```
<terminated> SwitchExample3 [Java Application] /
None of the above
```

Decision – switch (arrow in case)

- Java has introduced a new switch (arrow in case)
- Only available for Java 14 and above.
- No fall through, don’t need to use break.

```
 1 public class SwitchExample4 {
 2
 3     public static void main(String[] args) {
 4         int caseNum = 1;
 5
 6         switch(caseNum) {
 7             case 1 -> System.out.println("1");
 8             case 2 -> System.out.println("2");
 9             case 3 -> System.out.println("3");
10             default -> System.out.println("None of the above");
11         }
12
13     }
14
15 }
```

Output:

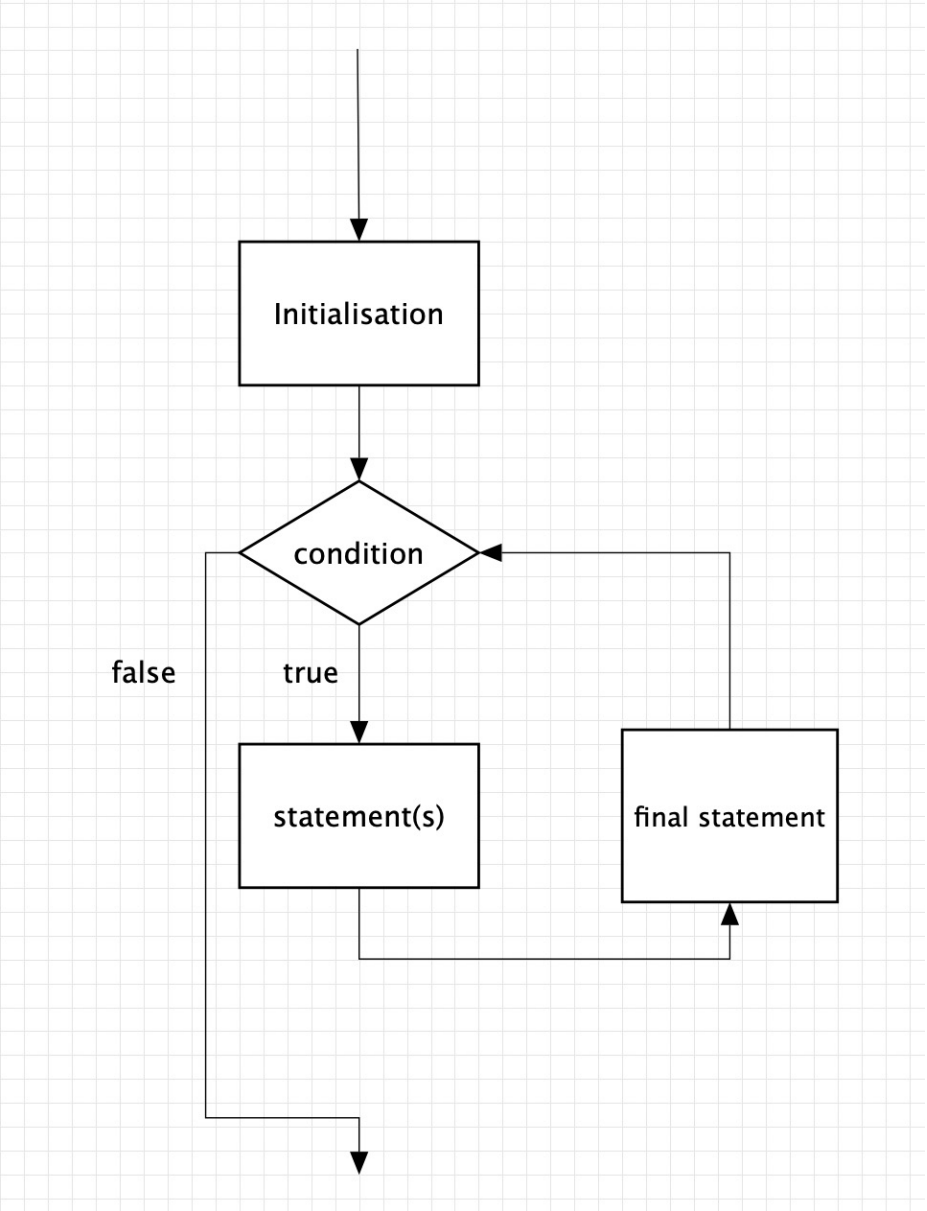


Repetition – for loop

- Syntax:

```
for (initialisation; terminating condition;
    statement(s)
}
```

```
1 public class ForLoopExample {
2
3     public static void main(String[] args) {
4         //print 0 to 9
5         for(int i=0;i<10;i++) {
6             System.out.println(i);
7         }
8     }
9 }
```

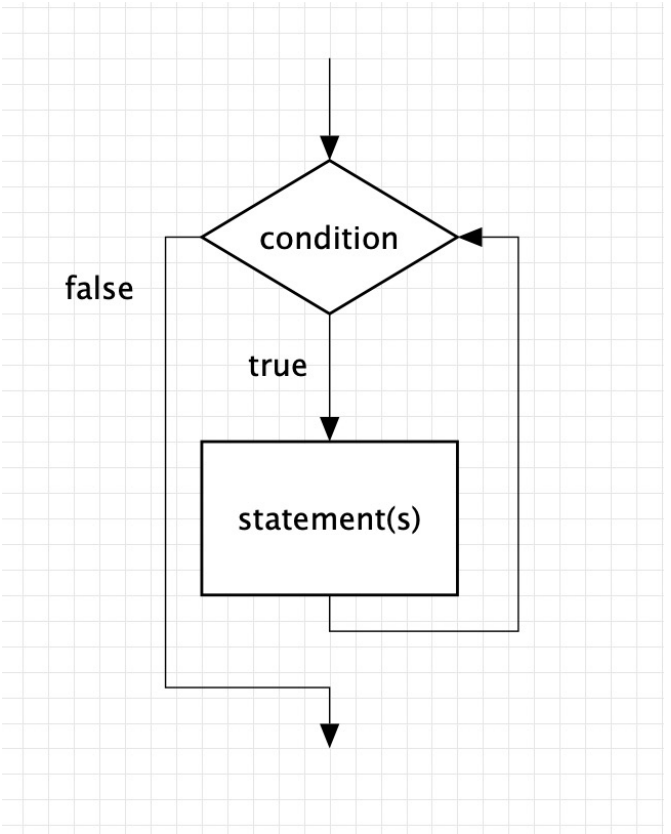


Repetition – while loop

- Syntax:

```
while (condition) {
    statement(s)
}
```

```
1 public class WhileLoopExample {
2
3     public static void main(String[] args) {
4         // TODO Auto-generated method stub
5         int i = 0;
6         while(i<10) {
7             System.out.println(i);
8             i+=1;
9         }
10    }
11 }
```



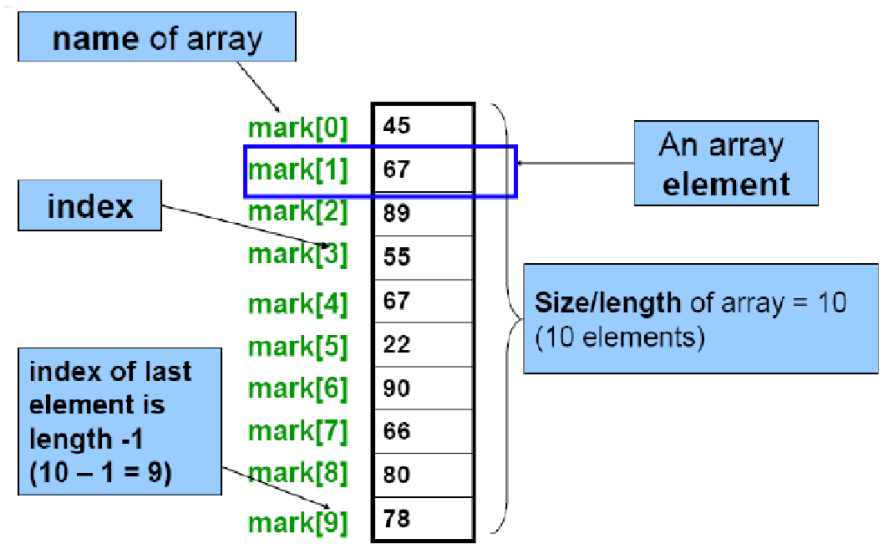
Repetition – infinite loop

- Be careful of infinite loop

```
1 public class InfiniteLoop {
2     public static void main(String[] args) {
3         int i = 0;
4         while(i<10) {
5             System.out.println(i);
6             //forgot to increment i
7         }
8     }
9 }
```

Array

- An array is a fixed-length sequence of values of the same type
- The type can be an object or a primitive type
- Size / length of array determines the number of items it contains.
- Each item is called an ELEMENT.
  - Size / length of array determines the number of elements it has.
- Elements of an array occupy consecutive memory space.
- Each element is accessed by its unique position number, also known as the INDEX or subscript).
  - Index always start from ZERO.
- Valid index is from zero to (length of array -1)



Array – Declaring and creating

- Syntax
- Declaring Array:
  - *data_type[] array_name;*
- Creating Array:
  - *array_name= new data_type[array_size];*
- *Declaring and creating Array:*
  - *data_type[] array_name= new data_type[array_size];*

```
1  public class ArrayExample1 {
2
3      public static void main(String[] args) {
4          //declare the array
5          int[] marks;
6
7          //create the array
8          marks = new int[5];
9
10         //declare and create the array
11         String[] names = new String[5];
12     }
13 }
```

Array – Initializing

- Using index

```java
public class ArrayExample1 {

    public static void main(String[] args) {
        //declare the array
        int[] marks;

        //create the array
        marks = new int[5];

        //declare and create the array
        String[] names = new String[5];

        marks[0] = 45;
        marks[1] = 67;
        marks[2] = 89;
        marks[3] = 77;
        marks[4] = 89;

        names[0] = "Peter";
        names[1] = "June";
        names[2] = "Marilyn";
        names[3] = "John";
        names[4] = "Bob";
    }
}
```

Array – Declaring, creating and Initializing

```
1  public class ArrayExample2 {
2
3      public static void main(String[] args) {
4          //declare, create and init the array
5          int[] marks = {45,67,89,77,89};
6
7          //declare and create the array
8          String[] names =  {"Peter","June","Marilyn","John","Bob"};
9      }
10 }
```

Array – Accessing Element

- Using index

```
1  public class ArrayExample2 {
2
3      public static void main(String[] args) {
4          //declare, create and init the array
5          int[] marks = {45,67,89,77,89};
6
7          //declare and create the array
8          String[] names =  {"Peter","June","Marilyn","John","Bob"};
9
10         System.out.println(names[0]+ ":" + marks[0]);
11         System.out.println(names[1]+ ":" + marks[1]);
12         System.out.println(names[2]+ ":" + marks[2]);
13         System.out.println(names[3]+ ":" + marks[3]);
14         System.out.println(names[4]+ ":" + marks[4]);
15
16     }
17 }
```

Array – using loop

```
1  public class ArrayExample2 {
2
3      public static void main(String[] args) {
4          //declare, create and init the array
5          int[] marks = {45,67,89,77,89};
6
7          //declare and create the array
8          String[] names =  {"Peter","June","Marilyn","John","Bob"};
9
10         for(int i=0;i<marks.length;i++) {
11             System.out.println(names[i]+ ":" + marks[i]);
12         }
13     }
14 }
```