Practical Computer Science 2
02-Review

Sayed Ahmad Sahim

Kandahar University
Computer Science Faculty

csadjava@gmail.com

February 11, 2018

Conttents

- Introduction to Java
- Data types
- Controle statements
- Input
- OOP

What is Java?

Introduction to Java

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible.(Gosling, James)

Java Editions

- Java Platform, **Standard Edition** or Java SE is a widely used computing platform for development and deployment of portable code for desktop and server environments.
- Java Platform, **Enterprise Edition** (Java EE), formerly known as Java 2 Platform, Enterprise Edition (J2EE), is a computing platform for development and deployment of enterprise software (network and web services).
- Java Platform, **Micro Edition** or Java ME is a computing platform for development and deployment of portable code for embedded and mobile devices (micro-controllers, sensors, gateways, mobile phones, personal digital assistants, TV set-top boxes, printers).

How to get a working environment for Java?

JVM

- JVM is an abstract computing machine, having an instruction set that uses memory. Virtual machines are often used to implement a programming language.
- JVM is the cornerstone of the Java programming language. It is responsible for Java's cross-platform portability and the small size of its compiled code.

- The Java Development Kit (JDK) is a software development environment used for developing Java applications and applets.
- It includes:
  - the Java Runtime Environment (JRE)
  - an interpreter/loader (java)
  - a compiler (javac)
  - an archiver (jar)
  - a documentation generator (javadoc)
  - other tools needed in Java development

Syntax of a Language

The set of rules you must follow when writing program code are referred to as syntax. in the first java hello world program we can see the following.

Syntax of Java

Code you saw in first java hello world program included the following parts

- Comments start with /* */ statements inside these signs are ignored by compiler
- The next line of the code was *public class HelloWorld* You must define a class in every Java program
- Class is a group of functions and characteristics in Java code that can be reused.

Continue

- Methods are groups of statements that are executed by your computer when instructed to do so.
- The next line of code starts the main() method definition *public static void main(String args[])*
- The next line of the code instructs computer to print Hello, world on the screen *System.out.println("Hello, world!");*
- *semicolon ;* are used to end statements like we use full stop in our languages to end a statement.
- The last two lines of code are the closing braces for the main() method and the HelloWorld class definition.

Comments

Adding comments to your code is not required, although it is definitely a good practice. Comments help you or anyone reading your code understand what your program does. Comments are not executed by the compiler.

We have two types of comment in java.

- Single Line starts with // until the end of the line.
- Multi Line comment starts /* */

The main() Method

When running a Java application, the main() method is the first thing the interpreter looks to. It acts as a starting point for your program and continues to drive it until it completes.
Every Java application requires a main() method or it will not run *public static void main(string args[])*.

- public makes this method accessible from other classes
- static ensures that there is only one reference to this method used by every instance of this program (class)
- main is the name of the method
- in front of the main method inside parenthesis are the arguments which the method accepts.
- String args[] Specifically, it is an array (list) of command-line arguments

Integrated Development Environment IDE

An integrated development environment (IDE) is a programming environment that has been packaged as an application program, typically consisting of a code editor, a compiler, a debugger, and a graphical user interface (GUI) builder.

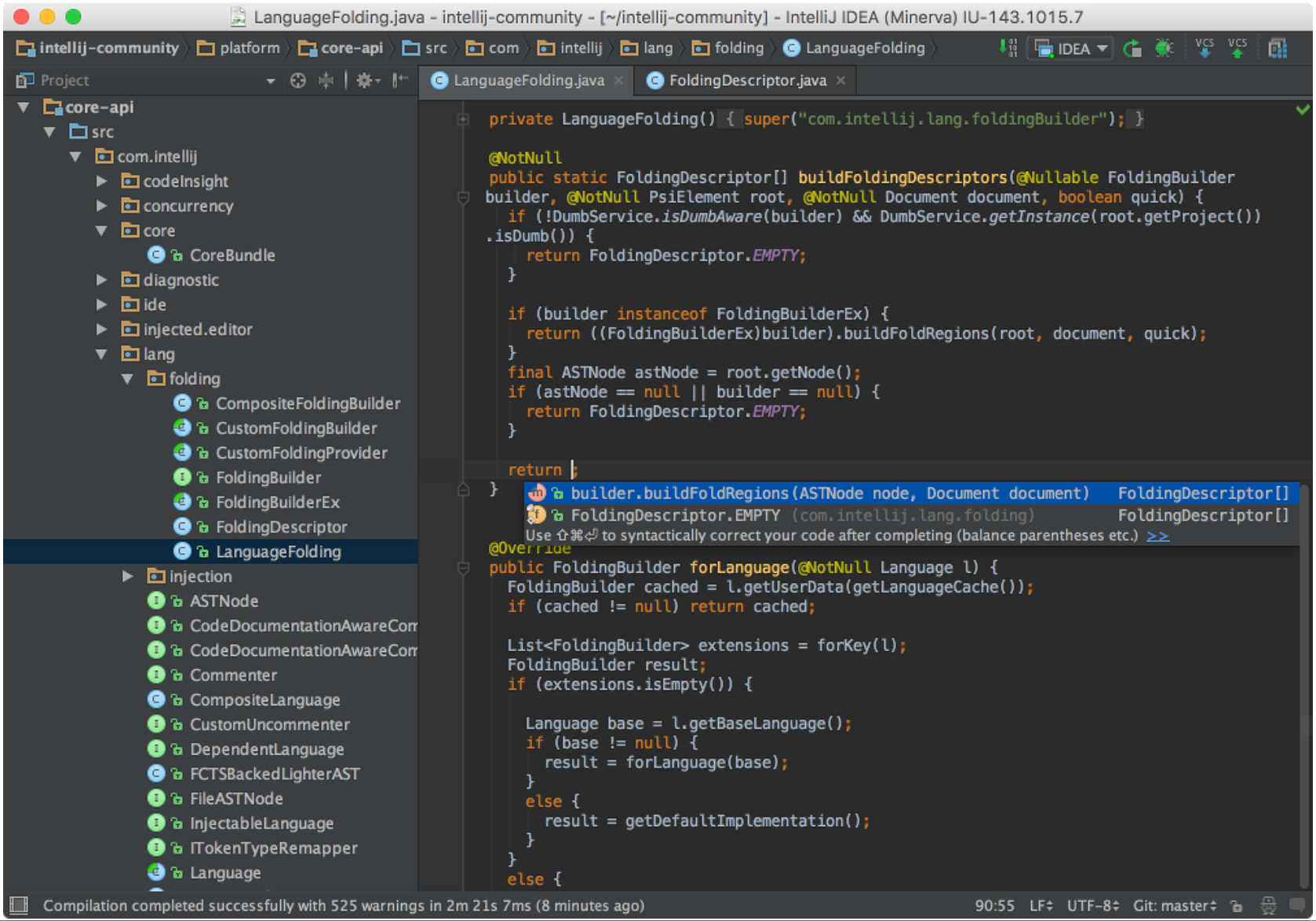
Question

Why should we use IDE?

Benefits of using IDE

- Compiler included with IDE
- Text editor is also Included with IDE
- Error checking
- Code navigation
- Code completion
- Code generation
- Code coloring
- Debugging

Intellij IDEA



Sublime Text

```java
public class JavaObject {

    private int num;

    public JavaObject(){
        num = 0;
    }

    public int getNum(){
        return num;
    }
    public void setVar(int number){
        num = number;
    }
}
```

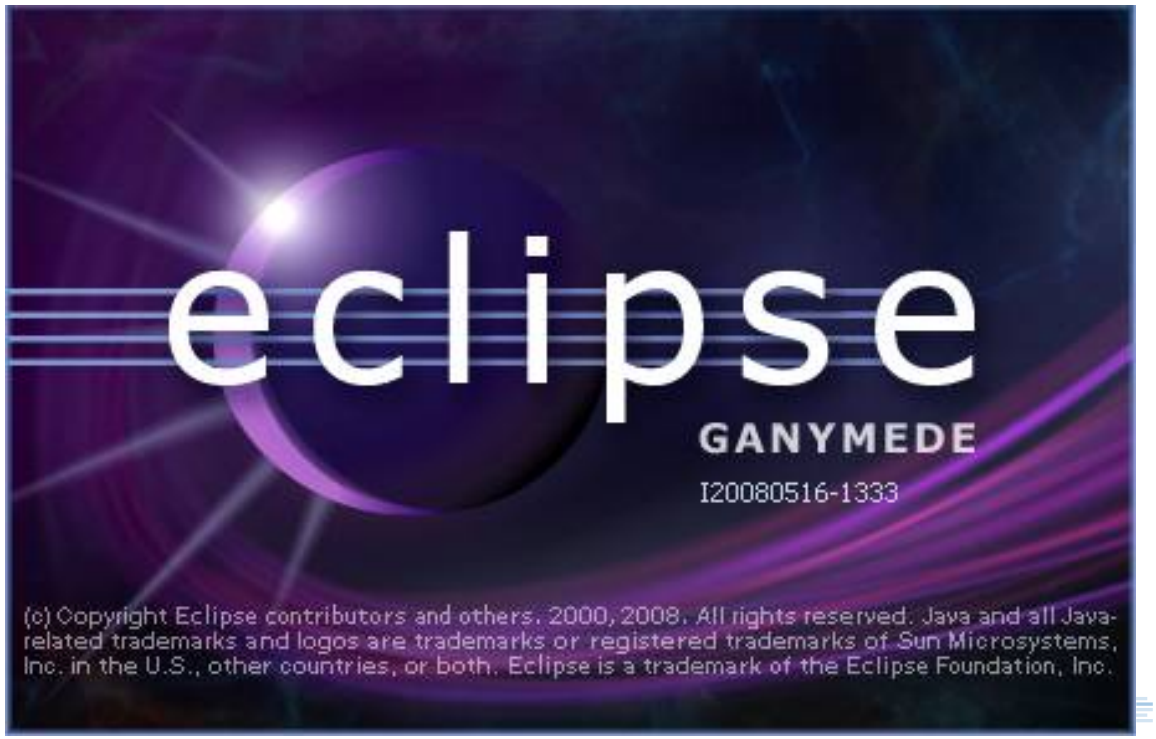
```java
public class Driver {
    public static void main(String[] args) {
        JavaObject obj = new JavaObject;

        obj.
    }
}
```

```
getNum()
setVar()
JavaObject()
```

Eclipse

Eclipse is a Java-based open source platform that allows a software developer to create a customized development environment (IDE) from plug-in components built by Eclipse members. Eclipse is managed and directed by the Eclipse.org Consortium.



NetBeans

NetBeans is an open-source integrated development environment (IDE) for developing with Java, PHP, C++, and other programming languages

- Open source Free IDE

Data types

- A data type is a set of values and a set of operations defined on them.
- For example, we are familiar with numbers and with operations defined on them such as addition and multiplication.
- There are two different types of variable:
  - Primitive/Builtin Data types
  - Referenced or User defined data types

Primitive Data type

Type	Description	Default	Size	Example Literals
boolean	true or false	false	1 bit	`true, false`
byte	twos complement integer	0	8 bits	(none)
char	Unicode character	`\u0000`	16 bits	`'a', '\u0041', '\101', '\\', '\'', '\n', 'ß'`
short	twos complement integer	0	16 bits	(none)
int	twos complement integer	0	32 bits	`-2, -1, 0, 1, 2`
long	twos complement integer	0	64 bits	`-2L, -1L, 0L, 1L, 2L`
float	IEEE 754 floating point	0.0	32 bits	`1.23e100f, -1.23e-100f, .3f, 3.14F`
double	IEEE 754 floating point	0.0	64 bits	`1.23456e300d, -1.23456e-300d, 1e1d`

Java Tokens

- A token is the smallest element in a program that is meaningful to the compiler. These tokens define the structure of the language.
- The Java token set can be divided into five categories:
  - Identifiers
  - Keywords
  - Literals
  - Operators
  - Separators.

Variables

- Instance Variables (Non-Static Fields)
- Class Variables (Static Fields)
- Local Variables. A method stores its temporary state in
- local variables.
- Parameters. They are the variables that are passed to the methods of a class.

Variable Declaration

- Identifiers are the names of variables.
- They must be composed of only letters, numbers, the underscore, and the dollar sign ($). They cannot contain white spaces.
- Identifiers may only begin with a letter, the underscore, or the dollar sign. A variable cannot begin with a number.
- All variable names are case sensitive.
- **Syntax** *datatype1 variable1, datatype2 variable2, datatypen variablen;*
- **Initialisation** *Variablename = value;*

Arrays

An array is a group of variables that share the same data type, and are referred to by a common name. Arrays of any type can be created and may have one or more dimensions.

Declaring Array Variables

- Arrays are indexed based data structures
- Size should be defined during declaration
- *Datatype[] name = initialization*

Well known operators

- Arithmetic Operators
- Assignment Operators
- Unary Operators
- Relational Operators
- Logical Operators

Arithmetic Operators

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra.

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0
++	Increment - Increases the value of operand by 1	B++ gives 21
--	Decrement - Decreases the value of operand by 1	B-- gives 19

Example

```
class OperatorExampleArithmetic{
    public static void main(String args[]){
            int a=10;
            int b=5;
            System.out.println(a+b);
            System.out.println(a-b);
            System.out.println(a*b);
            System.out.println(a/b);
            System.out.println(a%b);
        }
}
```

Unary Operators

- Postfix
- Prefix

```
class OperatorExample{
    public static void main(String args[]){
        int x=10;
        System.out.println(x++);
        System.out.println(++x);
        System.out.println(x--);
        System.out.println(--x);
    }
}
```

Unary Operators

```
class OperatorExample{
    public static void main(String args[]){
        int a=10;
        int b=10;
        System.out.println(a++ + ++a);
        System.out.println(b++ + b++);

        }
}
```

Relational Operators:

The equality and relational operators determine if one operand is greater than, less than, equal to, or not equal to another operand. The majority of these operators will probably look familiar to you as well. Keep in mind that you must use "==", not "=", when testing if two primitive values are equal.

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Bitwise Operators

The Java programming language also provides operators that perform bitwise and bit shift operations on integral type.

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>>	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

Example 1

```java
class OperatorExample2{
    public static void main(String args[]){
        int a=10;
        int b=-10;
        boolean c=true;
        boolean d=false;
        System.out.println(~a);//-11 (minus of total positive value which starts from
        System.out.println(~b);//9 (positive of total minus, positive starts from 0)
        System.out.println(!c);//false (opposite of boolean value)
        System.out.println(!d);//true
        }
}
```

Example 2

```
class OperatorExampleBitwise{
    public static void main(String args[]){
            System.out.println(10<<2);
            System.out.println(10<<3);
            System.out.println(20<<2);
            System.out.println(15<<4);

        }
}
```

Logical Operators

Logical Operators return boolean value of true or false.

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Assignment Operators

Assignment Operators are used to assign a value to a variable.

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator	C |= 2 is same as C = C | 2

Conditional Operator (? :)

- Conditional operator consists of three operands.
- After evaluating the boolean expression one of the value is assigned to the variable.
- variable x = (expression) ? value if true : value if false

```
public class Test {

   public static void main(String args[]){
      int a , b;
      a = 10;
      b = (a == 1) ? 20: 30;
      System.out.println( "Value of b is : " +  b );

      b = (a == 10) ? 20: 30;
      System.out.println( "Value of b is : " + b );
   }
}
```

Type Conversion in Java

- When you assign value of one data type to another, the two types might not be compatible with each other.
- If the data types are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion.
- If not then they need to be casted or converted explicitly.
- For example, assigning an int value to a long variable.

Widening or Automatic Type Conversion

Widening conversion takes place when two data types are automatically converted. This happens when:

- The two data types are compatible.
- When we assign value of a smaller data type to a bigger data type.
- For Example, in java the numeric data types are compatible with each other but no automatic conversion is supported from numeric type to char or boolean

Byte –> Short –> Int –> Long – > Float –> Double

Widening or Automatic Conversion

Example

```
class automaticTypeConversion
{
    public static void main(String[] args)
    {
        int i = 100;

        //automatic type conversion
        long l = i;

        //automatic type conversion
        float f = l;
        System.out.println("Int value "+i);
        System.out.println("Long value "+l);
        System.out.println("Float value "+f);
    }
}
```

Explicit Type Conversion

- If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing
  - This is useful for incompatible data types where automatic conversion cannot be done.
  - Here, target-type specifies the desired type to convert the specified value to.

Double –> Float –> Long –> Int –> Short –> Byte

Narrowing or Explicit Conversion

Example

```java
//Java program to illustrate explicit type conversion
class ExplicitTypeConversion
{
    public static void main(String[] args)
    {
        double d = 100.04;

        //explicit type casting
        long l = (long)d;

        //explicit type casting
        int i = (int)l;
        System.out.println("Double value "+d);

        //fractional part lost
        System.out.println("Long value "+l);

        //fractional part lost
        System.out.println("Int value "+i);
    }
}
```

Decision Making Statements

Control Flow Statements

- Normally statements in Java source code is using top-bottom approach.
- Control flow statements breakup the flow of execution by using conditional statements.
- Control flow statements:
  - Decision Making
  - Looping
  - Branching

if-then Statement

- if-then statement is the most basic of all the control flow statements.
- It tells your program to execute a certain section of code only if a particular test evaluates to true.

```
void applyBrakes() {
    // the "if" clause: bicycle must be moving
    if (isMoving){
        // the "then" clause: decrease current speed
        currentSpeed--;
    }
}
```

if-then-else Statement

- if-then-else statement provides a secondary path of execution when an "if" clause evaluates to false.
- You could use an if-then-else statement in the applyBrakes method to take some action if the brakes are applied when the bicycle is not in motion.

```
void applyBrakes() {
    if (isMoving) {
        currentSpeed--;
    } else {
        System.err.println("The bicycle has already stopped!");
    }
}
```

if...else if...else Statement:

- An if statement can be followed by an optional else if...else statement.
- which is very useful to test various conditions using single if...else if statement.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of the remaining else if's or else's will be tested.

```
public class Test {

   public static void main(String args[]){
      int x = 30;

      if( x == 10 ){
         System.out.print("Value of X is 10");
      }else if( x == 20 ){
         System.out.print("Value of X is 20");
      }else if( x == 30 ){
         System.out.print("Value of X is 30");
      }else{
         System.out.print("This is else statement");
      }
   }
}
```

Nested if...else Statement

- It is always legal to nest if-else statements.
- which means you can use one if or else if statement inside another if or else if statement.

```
public class Test {

   public static void main(String args[]){
      int x = 30;
      int y = 10;

      if( x == 30 ){
         if( y == 10 ){
            System.out.print("X = 30 and Y = 10");
         }
      }
   }
}
```

Switch

- Unlike if-then and if-then-else statements, the switch statement can have a number of possible execution paths.
- A switch works with the byte, short, char, and int primitive data types.

```
public class SwitchDemo {
    public static void main(String[] args) {

        int month = 8;
        String monthString;
        switch (month) {
            case 1:  monthString = "January";
                     break;
            case 2:  monthString = "February";
                     break;
            case 3:  monthString = "March";
                     break;
            case 4:  monthString = "April";
                     break;
            case 5:  monthString = "May";
                     break;
            case 6:  monthString = "June";
                     break;
            case 7:  monthString = "July";
                     break;
            case 8:  monthString = "August";
                     break;
            case 9:  monthString = "September";
                     break;
            case 10: monthString = "October";
                     break;
            case 11: monthString = "November";
                     break;
            case 12: monthString = "December";
                     break;
            default: monthString = "Invalid month";
                     break;
        }
        System.out.println(monthString);
    }
}
```

Switch Example 2

```java
public class Test {

   public static void main(String args[]){
      //char grade = args[0].charAt(0);
      char grade = 'C';

      switch(grade)
      {
         case 'A' :
            System.out.println("Excellent!");
            break;
         case 'B' :
         case 'C' :
            System.out.println("Well done");
            break;
         case 'D' :
            System.out.println("You passed");
         case 'F' :
            System.out.println("Better try again");
            break;
         default :
            System.out.println("Invalid grade");
      }
      System.out.println("Your grade is " + grade);
   }
}
```

Looping

Looping

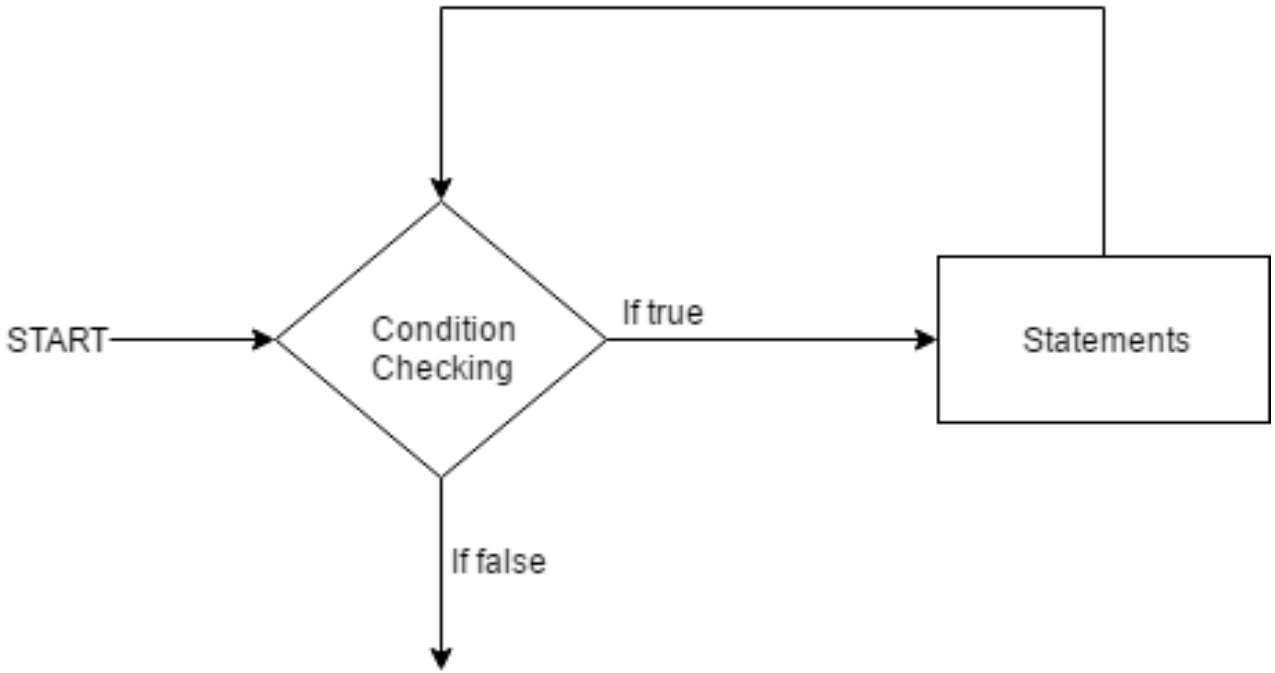
- Looping in programming languages is a feature which facilitates the execution of a set of instructions/functions repeatedly while some condition evaluates to true.
- Java provides three ways for executing the loops:
  1. For
  2. while
  3. do-while

while loop:

- A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition.
- The while loop can be thought of as a repeating if statement.

```
class whileLoopExample{
    public static void main(String[] args)
    {
        while (boolean condition){
            loop statements...
        }
    }
}
```

While loop flow chart



Example

```
class whileLoopExample{
    public static void main(String[] args)
    {

        int x = 1;

        // Exit when x becomes greater than 4
        while (x <= 4){
            System.out.println("Value of x:" + x);

            //increment the value of x for next itera
            x++;
        }
    }
}
```

for loop

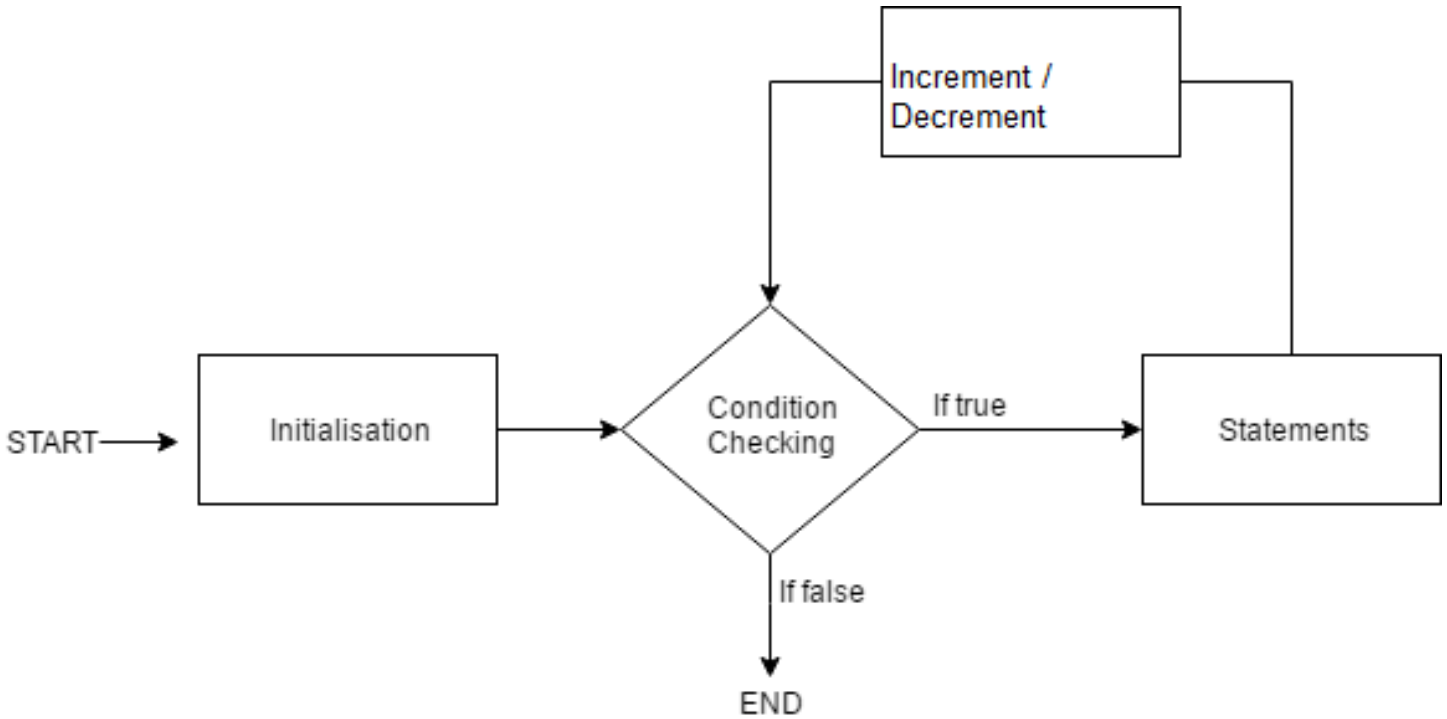
- for loop provides a concise way of writing the loop structure.
- Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

Syntax:

```
class forLoopExample{
    public static void main(String[] args)
    {

        for (initialization condition; testing condition; increment/decre
            {
                statement(s)
            }
    }
}
```

for-loop flow chart



Example

```
class forLoopExample{
    public static void main(String args[])
    {
        // for loop begins when x=2
        // and runs till x <=4
        for (int x = 2; x <= 4; x++)
            System.out.println("Value of x:" + x);
    }
}
```

Enhanced for loop

- Java also includes another version of for loop introduced in Java 5.
- Enhanced for loop provides a simpler way to iterate through the elements of a collection or array.
- It is inflexible and should be used only when there is a need to iterate through the elements in sequential manner without knowing the index of currently processed element.
- Syntax:

```
for (T element:Collection obj/array)
{
    statement(s)
}
```

Example

- Print all the elements of an array with for and enhanced for loop.

```java
// Java program to illustrate enhanced for loop
public class enhancedForLoop
{
    public static void main(String args[])
    {
        String array[] = {"Ron", "Harry", "Hermoine"};

        //enhanced for loop
        for (String x:array)
        {
            System.out.println(x);
        }

        /*for loop for same function
        for (int i = 0; i < array.length; i++)
        {
            System.out.println(array[i]);
        }
        */
    }
}
```

do while

- do while loop is similar to while loop with only difference that it checks for condition after executing the statements.
- It is an example of Exit Control Loop.
- It is important to note that the do-while loop will execute its statements at least once before any condition is checked, and therefore is an example of exit control loop.
- Syntax:

```
do
{
    statements..
}
while (condition);
```

Example

- do while loop starts with the execution of the statement(s). There is no checking of any condition for the first time.
- After the execution of the statements, and update of the variable value, the condition is checked for true or false

```
// Java program to illustrate do-while loop
class dowhileloopDemo
{
    public static void main(String args[])
    {
        int x = 21;
        do
        {
            //The line while be printer even
            //if the condition is false
            System.out.println("Value of x:" + x);
            x++;
        }
        while (x < 20);
    }
}
```

Pitfalls of Loops

- Infinite loop: One of the most common mistakes while implementing any sort of looping is that that it may not ever exit, that is the loop runs for infinite time.
- This happens when the condition fails for some reason.
- Syntax:

```
do
{
    statements..
}
while (condition);
```

Example 1

```java
//Java program to illustrate various pitfalls.
public class LooppitfallsDemo
{
    public static void main(String[] args)
    {

        // infinite loop because condition is not apt
        // condition should have been i>0.
        for (int i = 5; i != 0; i -= 2)
        {
            System.out.println(i);
        }
        int x = 5;

        // infinite loop because update statement
        // is not provided.
        while (x == 5)
        {
            System.out.println("In the loop");
        }
    }
}
```

Example 2

- Another pitfall is that you might be adding something into you collection object through loop and you can run out of memory.
- If you try and execute the below program, after some time, out of memory exception will be thrown.
- Syntax:

```java
//Java program for out of memory exception.
import java.util.ArrayList;
public class Integer1
{
    public static void main(String[] args)
    {
        ArrayList<Integer> ar = new ArrayList<>();
        for (int i = 0; i < Integer.MAX_VALUE; i++)
        {
            ar.add(i);
        }
    }
}
```

The continue Keyword

- The continue keyword can be used in any of the loop control structures.
- It causes the loop to immediately jump to the next iteration of the loop.

```
public class test {

   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         if( x == 30 ) {
          continue;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```

The break Keyword

- The break keyword is used to stop the entire loop.
- It is always used inside any loop or a switch statement.
- The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

```
public class test {

   public static void main(String args[]) {
      int [] numbers = {10, 20, 30, 40, 50};

      for(int x : numbers ) {
         if( x == 30 ) {
          break;
         }
         System.out.print( x );
         System.out.print("\n");
      }
   }
}
```

Methods

- A method is a piece of code that is called by a name that is associated with an object. In most respects it is identical to a function except for two key differences:
  1. A method is implicitly passed the object on which it was called.
  2. A method is able to operate on data that is contained within the class (remembering that an object is an instance of a class - the class is the definition, the object is an instance of that data).

Why method

- Method plays an important role in the conceptual design of a program.
- Any sequence of instructions that appears in a program more than once is a candidate for being made into a function which reduces the program size.
- Function code is stored once in memory and can be executed many times in the program.

Access Modifiers in java

- Java provides a number of access modifiers to set access levels for classes, variables, methods, and constructors.
- there are four access modifiers:
  1. Visible to the package, the default. No modifiers are needed.
  2. Visible to the class only (private).
  3. Visible to the world (public).
  4. Visible to the package and all subclasses (protected).

Declaring a Method

- Syntax:

```
modifier returnValueType methodName(list of parameters) {
  // Method body;
}
```

```
public static int funcName(int a, int b) {
  // body
}
```

1. public static : modifier
2. int: return type
3. funcName: function name
4. a, b: formal parameters
5. int a, int b: list of parameters

Example

```
public static int minFunction(int n1, int n2) {
    int min;
    if (n1 > n2)
        min = n2;
    else
        min = n1;

    return min;
}
```

Calling a Method

- Declaring a method, you define what the method is suppose to do.
- To use a method, you have to call or invoke it.
- There are two ways to call a method, depending on whether the method returns a value or not:
  1. If the method returns a value, a call to the method is usually treated as a value. For example *int larger = max(3, 4);* or *System.out.println(max(3, 4));*
  2. If the method returns void , a call to the method must be treated as a statement. *System.out.println("Welcome to Java!");*

Example

```
public class TestMax {
  /** Main method */
  public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i +
      " and " + j + " is " + k);
  }

  /** Return the max between two numbers */
  public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
      result = num1;
    else
      result = num2;

    return result;
  }
}
```

The void Keyword

The void keyword allows us to create methods which do not return a value. A call to a void method must be a statement.

```
public class ExampleVoid {

   public static void main(String[] args) {
      methodRankPoints(255.7);
   }

   public static void methodRankPoints(double points) {
      if (points >= 202.5) {
         System.out.println("Rank:A1");
      }
      else if (points >= 122.4) {
         System.out.println("Rank:A2");
      }
      else {
         System.out.println("Rank:A3");
      }
   }
}
```

Passing Parameters by Values

- Parameters are the variables provided to methods as an input data.
- While calling the method variables values should be provided in the same order as they are defined in method declaration.

For example, the following method prints a message n times:

```
public static void nPrintln(String message, int n) {
  for (int i = 0; i < n; i++)
    System.out.println(message);
}
```

Pass by value example

```
public class Increment {
  public static void main(String[] args) {
    int x = 1;
    System.out.println("Before the call, x is " + x);
    increment(x);
    System.out.println("after the call, x is " + x);
  }

  public static void increment(int n) {
    n++;
    System.out.println("n inside the method is " + n);
  }
}
```

Modularizing Code

- Methods can be used to reduce redundant code and enable code reuse.
- Methods can also be used to modularize code and improve the quality of the program.

```
import java.util.Scanner;
 public class GreatestCommonDivisorMethod {
/** Main method */
 public static void main(String[] args) {
        // Create a Scanner
        Scanner input = new Scanner(System.in);
        // Prompt the user to enter two integers
        System.out.print("Enter first integer: ");
        int n1 = input.nextInt();
        System.out.print("Enter second integer: ");
        int n2 = input.nextInt();
        System.out.println("The greatest common divisor for " + n1 +
        " and " + n2 + " is " + gcd(n1, n2) );
}
/** Return the gcd of two integers */
public static int gcd(int n1, int n2) {
  int gcd = 1; // Initial gcd is 1
  int k = 2; // Possible gcd
  while (k <= n1 && k <= n2) {
   if (n1 % k == 0 && n2 % k == 0)
   gcd = k; // Update gcd
   k++;
 }
return gcd; // Return gcd
}
}
```

Advantages of using modularization

- By encapsulating the code for obtaining the gcd in the previous method can have several advantages:
  1. It isolates the problem for computing the gcd from the rest of the code in the main method. Thus, the logic becomes clear and the program is easier to read.
  2. The errors on computing gcd are confined in the gcd method, which narrows the scope of debugging.
  3. The gcd method now can be reused by other programs.

Overloading Methods

- When a class has two or more methods by same name but different parameters, is known as method overloading.
- Method overloading is different from method overriding.
- In overriding a method has same method name, type, number of parameters etc.

Method overloading

Example

Lets consider the example shown before for finding minimum integer number. lets say we want to find minimum number of double type.

```
public class ExampleOverloading{

   public static void main(String[] args) {
      int a = 11;
      int b = 6;
      double c = 7.3;
      double d = 9.4;
      int result1 = minFunction(a, b);
      // same function name with different parameters
      double result2 = minFunction(c, d);
      System.out.println("Minimum Value = " + result1);
      System.out.println("Minimum Value = " + result2);
   }
```

```
// for integer
 public static int minFunction(int n1, int n2) {
    int min;
    if (n1 > n2)
       min = n2;
    else
       min = n1;

    return min;
 }
 // for double
 public static double minFunction(double n1, doul
   double min;
    if (n1 > n2)
       min = n2;
    else
       min = n1;

    return min;
 }
```

Variables Scope

- The scope of a variable defines the section of the code in which the variable is visible.
- As a general rule, variables that are defined within a block are not accessible outside that block.
- The lifetime of a variable refers to how long the variable exists before it is destroyed.

- **Instance variables** Instance variables are those that are defined within a class itself and not in any method or constructor of the class.
- **Argument variables** These are the variables that are defined in the header of a constructor or a method.
- **Local variables** A local variable is the one that is declared within a method or a constructor (not in the header). The scope and lifetime are limited to the method itself.

Variable Scope

It is fine to declare i in two nonnested blocks

```
public static void method1() {
  int x = 1;
  int y = 1;

  for (int i = 1; i < 10; i++) {
    x += i;
  }

  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```

It is wrong to declare i in two nested blocks

```
public static void method2() {

  int i = 1;
  int sum = 0;

  for (int i = 1; i < 10; i++)
    sum += i;
  }

}
```

Java built-in functions

- Built in functions in java are methods that are present in different API of JDK.
- For example cos(double a), exp(double a) etc are built in function of java present in java.lang.Math class.

The Math Class

The Math class contains the methods needed to perform basic mathematical functions. You have already used the pow(a, b) method to compute a^b

Trigonometric Methods

The Math class contains the following trigonometric methods

```
/** Return the trigonometric sine of an angle in radians */
public static double sin(double radians)

/** Return the trigonometric cosine of an angle in radians */
public static double cos(double radians)

/** Return the trigonometric tangent of an angle in radians */
public static double tan(double radians)

/** Convert the angle in degrees to an angle in radians */
public static double toRadians(double degree)

/** Convert the angle in radians to an angle in degrees */
public static double toDegrees(double radians)

/** Return the angle in radians for the inverse of sin */
public static double asin(double a)

/** Return the angle in radians for the inverse of cos */
public static double acos(double a)

/** Return the angle in radians for the inverse of tan */
public static double atan(double a)
```

Exponent Methods

There are five methods related to exponents in the Math class:

```
/** Return e raised to the power of x (e^x) */
public static double exp(double x)

/** Return the natural logarithm of x (ln(x) = log_e(x)) */
public static double log(double x)

/** Return the base 10 logarithm of x (log_10(x)) */
public static double log10(double x)

/** Return a raised to the power of b (a^b) */
public static double pow(double a, double b)

/** Return the square root of x (√x) for x >= 0 */
public static double sqrt(double x)
```

The Rounding Methods

The Math class contains five rounding methods:

```
/** x is rounded up to its nearest integer. This integer is
  * returned as a double value. */
public static double ceil(double x)

/** x is rounded down to its nearest integer. This integer is
  * returned as a double value. */
public static double floor(double x)

/** x is rounded to its nearest integer. If x is equally close
  * to two integers, the even one is returned as a double. */
public static double rint(double x)

/** Return (int)Math.floor(x + 0.5). */
public static int round(float x)

/** Return (long)Math.floor(x + 0.5). */
public static long round(double x)
```

Question