1. **Numbers: ( java.lang** package**)**

* Primitive Numbers – Most of the time we use the primitive type

int i = 500;

float gpa = 3.65;

byte mask = 0xff;

* Wrapper Classes – **Boxing** and **Unboxing**(Sometimes called **Autoboxing**)
  + Java platform provides wrapper classes for each of the primitive data types.
  + These classes "wrap" the primitive in an object.
  + Wrapping is done by the compiler
  + If you use a primitive where an object is expected, the compiler **boxes** the primitive in its wrapper class
  + Similarly, if you use a number object when a primitive is expected, the compiler **unboxes** the object
  + Example of **boxing** and **unboxing**:

Integer x, y;

x = 12;

y = 15;

System.out.println(x+y);

* All of the numeric wrapper classes are subclasses of the abstract class Number:

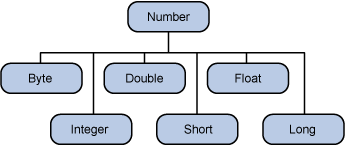


Fig - Source: developer.sun.com

* BigDecimal, BigInteger – High Precision
* AtomicInteger, and AtomicLong – multi – threaded applications

1. Int CompareTo()
2. boolean equals(Object obj)
3. static Integer decode(String s)
4. static int parseInt(String s)
5. static int parseInt(String s, int radix) – radix : 10, 2, 8, 16
6. String toString()
7. static Integer valueOf(int i)
8. static Integer valueOf(String s)
9. static Integer valueOf(String s, int radix)

**The printf() and format() Methods**

System.out.format("%d%n", n);

System.out.printf("%d%n", n);

### The DecimalFormat Class

* [java.text.DecimalFormat](http://java.sun.com/javase/6/docs/api/java/text/DecimalFormat.html) - Used to display of leading and trailing zeros, prefixes and suffixes, grouping (thousands) separators, and the decimal separator.
* **DecimalFormat myFormatter=new DecimalFormat(**"###,###.###",123456.789**);**

**myFormatter.format(value);**

**Math Class :** exp, log, pow, sqrt, sin, cos, tan, max, min, round, abs

**Random Numbers:** Math.Random() and java.util.Random

1. **Characters**

char ch = 'a';

char uniChar = '\u039A'; // Unicode for uppercase Greek omega

// an array of chars

character char[] charArray ={ 'a', 'b', 'c', 'd', 'e' };

1. **Strings - Immutable**

String greeting = "Hello world!";

String string = new String(“This is a String”);

Length – string.length()

Concatenate - string1.concat(string2);

"My name is ".concat("Ram");

My name is " + “Ram”;

String to Number – valueOf

Substring - String substring(int beginIndex, int endIndex)

String[] split(String regex)   
String[] split(String regex, int limit)

String trim()

String toLowerCase()   
String toUpperCase()

int indexOf(int ch)   
int lastIndexOf(int ch)

boolean contains(CharSequence s)

boolean endsWith(String suffix)  
boolean startsWith(String prefix)

int compareTo(String anotherString)

int compareToIgnoreCase(String str)

boolean equals(Object anObject)

boolean equalsIgnoreCase(String anotherString)

boolean matches(String regex)

**StringBuilder – mutable**

**LAB:** Write a program that computes your initials from your full name and displays them.

**Operators:**

* Arithmetic Operators : +, -, /, \*, %
* Unary Operators: +, -, ++, --, !
* Equality and Relational Operators : ==, !=, >, >=, <, <=
* Conditional Operators : && Conditional-AND, || Conditional-OR
* Type Comparison Operator : instanceOf

**Conditional Expressions:**

1. If – Then Statement – with or without braces
2. If – Then – Else Statement
3. Switch Statement **-** works with the byte, short, char, int, and enumerated primitive data types
4. While

while (expression) {

statement(s)

}

1. DO While

do {

statement(s)

} while (expression);

1. For

for (initialization; termination; increment) {

statement(s)

}

When using this version of the for statement, keep in mind that:

* The *initialization* expression initializes the loop; it's executed once, as the loop begins.
* When the *termination* expression evaluates to false, the loop terminates.
* The *increment* expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment *or* decrement a value.

The three expressions of the for loop are optional; an infinite loop can be created as follows:

for ( ; ; ) { // infinite loop

// your code goes here

}

1. Break Statement – labeled and unlabeled

**Labeled:**

class BreakWithLabelDemo {

public static void main(String[] args) {

int[][] arrayOfInts = { { 32, 87, 3, 589 },

{ 12, 1076, 2000, 8 },

{ 622, 127, 77, 955 }

};

int searchfor = 12;

int i;

int j = 0;

boolean foundIt = false;

search:

for (i = 0; i < arrayOfInts.length; i++) {

for (j = 0; j < arrayOfInts[i].length; j++) {

if (arrayOfInts[i][j] == searchfor) {

foundIt = true;

break search;

}

}

}

if (foundIt) {

System.out.println("Found " + searchfor +

" at " + i + ", " + j);

} else {

System.out.println(searchfor

+ " not in the array");

}

}

}

1. Continue
2. Return

**Expressions**

* Made up of variables, operators, and method invocations, which are constructed according to the syntax of the language that evaluates to a single value.

int **result = 1 + 2**;

* Compound Expression: 1\*2\*3, 4+6/7 – ambiguous, (4+8)/5 - unambiguous

**Statements**

* Statements are roughly equivalent to sentences in natural languages.
* A statement forms a complete unit of execution
* Terminated with semicolon (;)
  + Assignment expressions: aValue = 8933.234;
  + Any use of ++ or --: aValue++;
  + Method invocations: System.out.println("Hello World!");
  + Object creation expressions: Bicycle myBike = new Bicycle();

**Blocks**

* *block* is a group of zero or more statements between balanced braces

if (condition) { **// begin block 1**

System.out.println("Condition is true.");

} **// end block one**

else { **// begin block 2**

System.out.println("Condition is false.");

} **// end block 2**

**LAB: Write a program using a break, continue, and return statement.**

**Array**

* An *array* is a container object that holds a fixed number of values of a single type.
* The length of an array is established when the array is created. After creation, its length is fixed.
* Each item in an array is called an *element*, and each element is accessed by its numerical *index*
  + anArray = new int[10]; : Initialization
  + anArray[0] = 100; // initialize first element

OR

int[] anArray = {100, 200, 300, 400, 500, 600, 700, 800};

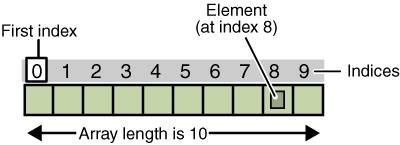
* Multidimensional Array : Array of Arrays

String[][] names = {{"Mr. ", "Mrs. ", "Ms. "},

{"Ram", "Sita"}};

System.out.println(names[0][0] + names[1][0]); //Mr. Ram

System.out.println(names[0][2] + names[1][1]); //Ms. Sita



**Fig: Java Tutorial (Oracle)**

* **Copy Array**

public static void main(String[] args) {

char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',

'i', 'n', 'a', 't', 'e', 'd' };

char[] copyTo = new char[7];

System.arraycopy(copyFrom, 2, copyTo, 0, 7);

System.out.println(new String(copyTo)); - caffein

}

**Variable**

* An object stores its state in *fields*.

int **cadence** = 0;  
int **speed** = 0;  
int **gear** = 1;

* In the Java programming language, the terms "field" and "variable" are actually both used
* **Instance Variables (Non-Static Fields)**
  + Objects actually store their individual state in "non-static fields", that is, fields declared without the static keyword.
  + Non-static fields are also known as *instance variables* because their values are unique to each *instance* of a class
* **Class Variables (Static Fields)**
  + A *class variable* is any field declared with the static modifier; this tells the compiler that there is exactly one copy of this variable in existence; regardless of how many times the class has been instantiated.
* **Local Variables**
  + Similar to how an object stores its state in fields
  + A method will often store its temporary state in *local variables*
* **Parameters**
  + Parameters are always classified as "variables" not "fields".
* **Scope of a Variable**
  + Class level, method level, block level

**Coding Conventions**

* **Why Have Code Conventions**
* Code conventions are important to programmers for a number of reasons:
  + 80% of the lifetime cost of a piece of software goes to maintenance.
  + Hardly any software is maintained for its whole life by the original author.
  + Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
  + If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.
* File Name: Java source .java, Java bytecode .class
* Naming Convention: Class: Camel Case(First alphabet caps), Interface: Camel Case(First alphabet caps), Methods: Camel Case(First alphabet small), Variables: Camel Case(First alphabet small), Constants: All Caps
* Variable declaration: int a, b; int a; int b;
* Proper use of public, private, and protected
* Parenthesis: if (a == b && c == d) // AVOID!

if ((a == b) && (c == d)) // RIGHT

* Maintaining java docs
* Proper use of logger

**LAB:**

1. Write a program using switch statement to develop a simple calculator for +, -, \*, /, and % operators.
2. Write a program that displays the temperatures from 0 degrees Celsius to 100 degrees Celsius and its Fahrenheit equivalent. Note that the conversion from Celsius to Fahrenheit uses the following formula: F = C \* 9/5 + 32;
3. A palindrome is a sequence of characters that reads the same backward as forward. For example, each of the following five-digit integers is a palindrome: 12321, 55555, 45554 and 11611. Write an application that reads in a five-digit integer and determines whether it is a palindrome. If the number is not five digits long, display an error message and allow the user to enter a new value.
4. Write a program to find the number of and sum of all integers greater than 100 and less than 200 that are divisible by 7.