Commands:

* Javac – compile java program
* Java – run java program
* Javap – get methods of a class

Packages:

* Scanner – java.util.Scanner;
  1. nextInt()
  2. nextFloat()
  3. nextByte()
  4. nextLine()
  5. nextDouble()
  6. hasNextInt()
  7. useRadix(num)
* String – java.lang.String;
  1. length()
  2. toLowerCase()
  3. toUpperCase()
  4. trim()
  5. substring(int begin,int end)
  6. replace(char old, char new)
  7. startsWith(String s)
  8. endsWith(String s)
  9. equals(String s)
  10. equalsIgnoreCase(String s)
  11. compareTo(String s)
  12. valueOf(int i)

Notes:

* Datatypes
  1. Primitive Datatypes
     + Integral
       - Byte - 1
       - Short - 2
       - Int - 4
       - Long - 8
     + Floating point
       - Float - 4
       - Double - 8
     + Char - 2
     + Boolean – Min 1 bit or Max 1 byte
* Variables – Naming Rules
  1. Case sensitive
  2. Contains alphabets, numbers , \_, $
  3. Starts with Alphabet, \_, $
  4. Should not be a keyword
  5. Should not be a class name, If class is also in use
  6. No limit on length of name
  7. Follow Camel Cases
* Literals
  1. 5 is an Integer literal
  2. 153.75 is double literal
  3. “Java” String literal
  4. ‘java’ character literal
  5. 5L long literal
  6. 2.5f float literal
  7. True/false Boolean literal
  8. To print a decimal number in binary or octal or hexadecimal form use method Integer.toBinaryString(x);
  9. Floating point numbers are represented as 16352E-2 which is 163.52 (16352 x 10-2)
  10. Characters are Represented as ASCII (American standard code for information interchange) codes
  11. Every language in the world has a code called as UNICODE
  12. ACSII is a subset of UNICODE
  13. Java uses UNICODE to support multiple languages that is the reason character size in java is 2 bytes
* Interpreter and Compiler- language can either be compiled or interpreted or combination of both called as hybrid language.
  1. Interpreter translates the code line by line
     + JavaScript is an interpreted language
  2. Compiler translates all at once
     + C/C++ are compiled language
  3. Java is a hybrid language that is compiled and interpreted







Execution Engine has JIT/Interpreter and Garbage collector

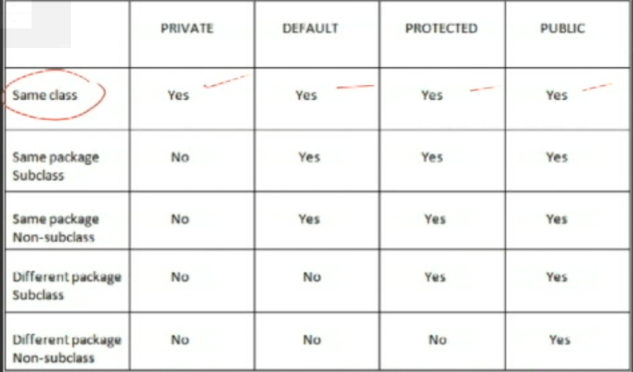
* Operators
  1. Increment/Decrement Operators
     + ++,--
  2. Arithmetic Operators
     + +, - , %, \*, /
  3. Bitwise Operators
     + &,|,~,^,<<,>>,>>>
  4. Relational Operators
     + <,>,<=,>=,!=,==
  5. Logical Operators
     + &&,||,!
* Coercion and Casting
  1. Type coercion is the automatic or implicit conversion of values from one data type to another Operators
  2. Type casting explicit conversion of values from one data type to another Operators
* Bitwise
  1. X=0b1000
  2. X<<2 = 32
  3. X>>2 = 2
  4. X>>1 = 4
  5. X>>>1 = 21471231
* Bit Masking and Bit Merging
* System.out.printf(“Hello”); -- same as c/c++ printf();
* System.out.printf(“%d”, a);
  1. %+d – shows the sign
  2. % (d – represents in brackets
  3. %1$d – argument index
  4. %6f – width
  5. %6.2f – number of places before and after decimal
  6. %20s – width
  7. %s20 – spaces are left aligned
* Regular Expressions







* Arrays:
  + Datatype [] arrayname = new Datatype[size]; -- one dimensional array
  + For(int X: in A){} – for each loop
  + Array initialization Datatype [] arrayname={};
  + Jagged array
    - Multidimensional array with each array of different size
    - Int A[][]
    - A=new int[3][];
    - A[0] = new int[2];
    - A[1]=new int[4];
    - A[2]=new int[3];
* Methods
* OOP
  + Principles of OOP
    - Abstraction
    - Encapsulation
    - Inheritance
      * Deriving a new class from existing class is known as inheritance
      * Specialization in java is achieved using inheritance
    - Polymorphism
* **Generalization** In Generalisation group of classes are referred with Super class with single name. Generalisation means **Bottom Up.** In Generalisation A Super Class Is made by Grouping Multiple Sub Classes. Generalisation is achieved using **Interfaces.**
* **Specialisation** a new Sub Class is Generated by borrowing the features of existing concrete class and adding new features to it. Specialisation means **Top Down.** In specialisation a new Class is derived from an existing Super Class. Specialisation is achieved using **Inheritance.**
* Dynamic method dispatch
  + Used to achieve runtime polymorphism
  + Super sup = new Subclass ();
  + Creating a subclass object with superclass reference
  + Methods of subclass Is called as methods are called based on objects
  + Only the methods in super class overloaded in subclass can be used
* Abstract Classes
  + An abstract class can be a super class but much of the class functionality maybe undefined or unimplemented. This gap has to be filled by the programmer in the subclass.
  + Abstract methods start with abstract keyword and the class that has at least one abstract method is an abstract class
  + A class can be an abstract class even if it does not have at least one abstract method;
* Interface is a special case of an abstract class, which contains all abstract methods. An interface specifies what a class must do but not how to do.
* Inner Class
  + Nested Inner Class
    - A class with in a class the inner class can be used after creating the object of the outer class first then inner class
    - Outer.Inner oi = new Outer().new Inner();
  + Local Inner Class
  + Anonymous Inner Class
    - Abstract class whose object can be created and it’s methods can be declared anonymously
    - abstract class My{
    - abstract public void show ();
    - }
    - class Outer1{
    - public void display (){
    - My m = new My(){
    - @Override
    - public void show(){
    - System.out.println("show");
    - }
    - };
    - m.show();
    - }
    - }
  + Static Inner Class
    - class Outer{
    - int x =10;
    - static int y =20;
    - static class My{
    - static void show(){
    - System.out.println(y);
    - }
    - }
    - }
    - //In main class
    - Outer.My m= new Outer.My();
    - Outer.My.show();
* Static and final
  + Static
    - Static keyword is used for representing the metadata
    - Static members are used for representing the information or data related to a class
    - There are static variables, methods, nested class, blocks
  + Final
    - Final variables are like constants that means we cannot change them later
    - Final classes cannot be inherited
    - Final members can be initialized directly or within instance block or inside constructor
* Singleton class
  + Only one object of the class can be created
  + It can be created by making the constructor private and returning the class object by calling the constructor from another method within the class
  + class CoffeeMachine{
  + private float coffeeQty;
  + private float milkQty;
  + private float sugarQty;
  + static private CoffeeMachine my = null;
  + private CoffeeMachine(){
  + coffeeQty=1;
  + milkQty=1;
  + sugarQty=1;
  + }
  + public float getCoffee(){
  + return 0.15f;
  + }
  + static CoffeeMachine getInstance()
  + {
  + if(my==null)
  + my = new CoffeeMachine();
  + return my;
  + }
  + }
* Packages
  + Collection of classes or other packages
  + Write java program with package package\_name as first statement;
  + Save the program
  + compile the program using javac -d . Demo.java
* Access Modifier
  + Default
    - Accessible in same package in all sub and non-sub classes
  + Public
    - Accessible in all packages sub and non-sub packages
  + Private
    - Accessible in only same class
  + Protected
    - Accessible in same class and all packages sub-classes



* Exception
  + An error that occurs during the execution of a program
  + Runtime errors are exceptions
  + Exception handling is done using try, catch, finally, throw and throws
  + Resource needs to be closed after use
  + Include the closing statements in the finally block
* Multiprogramming
  + Multi-user
  + Multi-Tasking
    - Multi-threading
      * multithreading can be achieved in java in two ways
        + By extending Thread class

Methods In thread class

Getters and setters

getId()

getName()

getPriority()

getState()

getThreadGroup()

setName(String name)

setPriority(int p)

setDaemon(Boolean d)

Enquiry

Boolean isAlive()

Boolean isDaemon()

Boolean isInterrupted()

Instance

Void interrupt()

Join()

Join(long millis)

Run()

Start()

Static methods

Int activeCount()

Thread currentThread()

Void yield()

Void dumpStack()

* + - * + By implementing Runnable interface
* Starvation
  + If some thread has higher priority then other threads with lower priority has to wait this is called starvation
* Synchronization
  + Synchronization concept
  + Resource sharing
  + Critical Section
    - A piece of program used by multiple threads repeatedly
  + Mutual Exclusion
  + Locking/Mutex
  + Semaphore
  + Monitor
  + Race Condition
  + Inter process communication
* Object class
  + Methods of object class
    - Hashcode()
    - Wait()
    - Notify()
    - notifyAll()
    - equals()
    - toString()
    - getClass()
  + We can override all object methods
* Wrapper classes
  + Object class
    - Number class
      * Integer
      * Byte
      * Short
      * Integer
      * Long
      * Float
      * Double
      * Boolean
    - Character
    - Boolean
* String, StringBuffer and StringBuilder
  + String
  + StringBuffer
    - Mutable
    - Thread-safe – cannot be used by two threads at the same time
  + StringBuilder
    - Mutable
    - Not Thread-safe
* Java.io
  + Class InputStream
    - Int read()
    - Int read(byte[] b)
    - Int read(byte[] b,int off,int len)
    - Int available()
    - Long skip(long n)
    - Void mark(int limit)
    - Void reset()
    - Boolean markSupported()
    - Void close()
  + Class OutputStream
    - Void write()
    - Void write(byte[])
    - Void flush()
    - Void close()
  + Sequence input stream() – read from multiple files
  + RandomAccessFile()