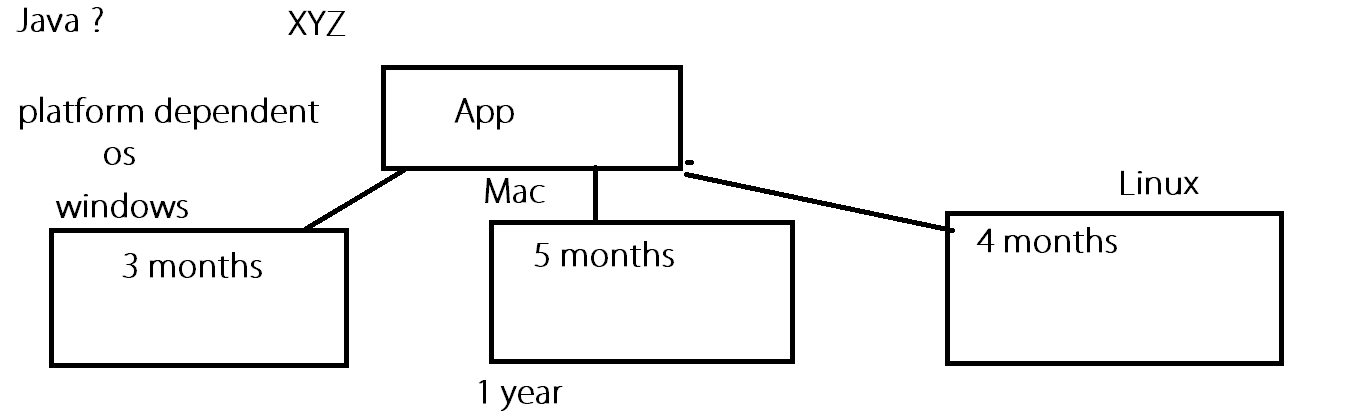
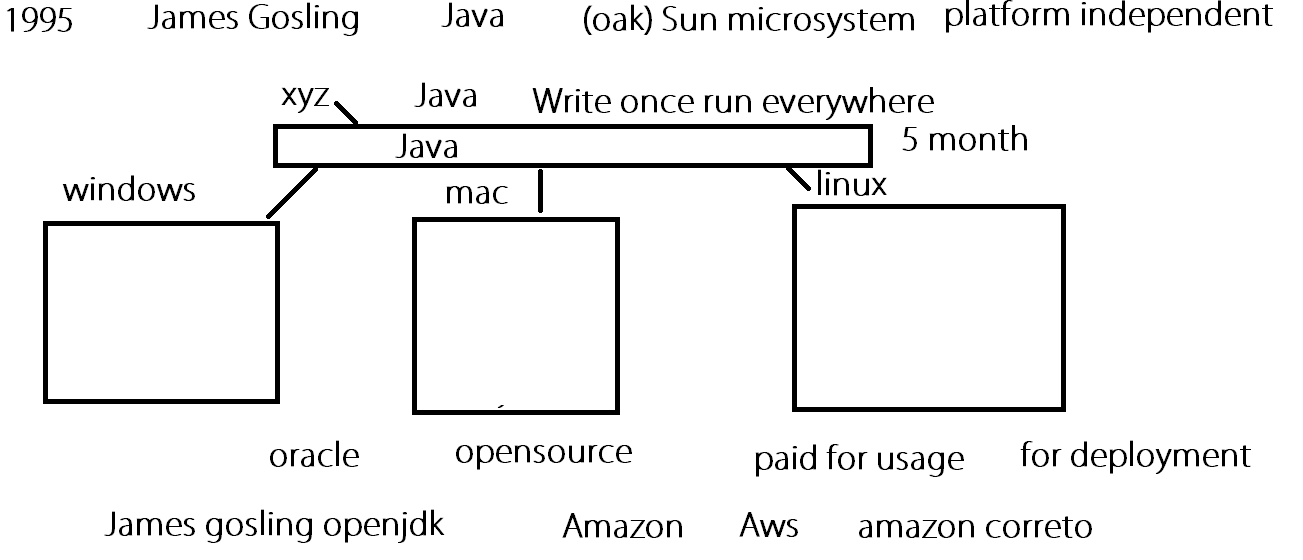
Object oriented programming

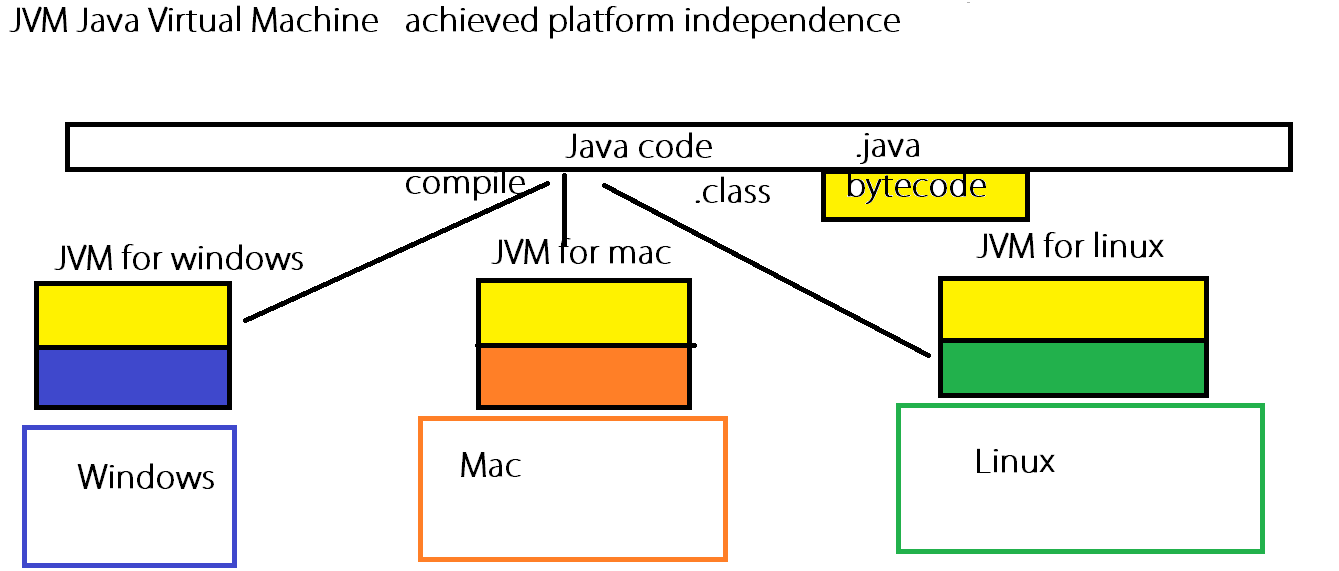
OOP is programming language used to deal with real world problems or entities. This focuses on parameters as well as methods (function).

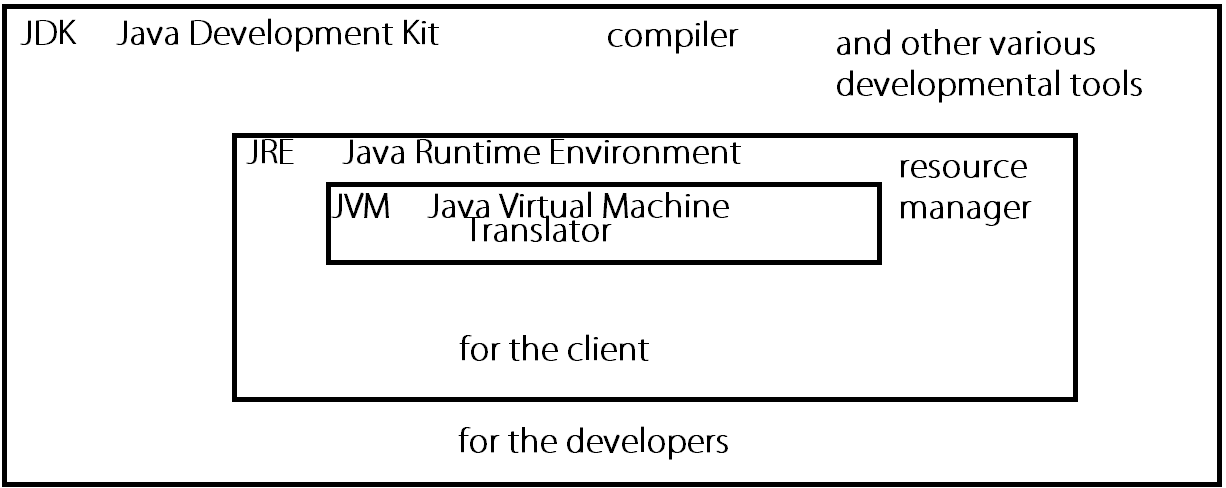
OOP feature:

* Classes - blueprint/map logical map | it does not occupy memory
* Object - physical manifestation of the classes. It occupies memory
* Inheritance – transfer of properties from parent to child classes
* Polymorphism – different forms of classes and object’s properties based on environment
* Encapsulation – data hiding- secure the object’s properties by wrapping them
* Abstraction – feature hiding – hide unnecessary work detail from others. Hide the working mechanism of method/function and only supply its name.









Java features:

* Object Oriented Program
* Platform independence
* Simple Language | Easy to learn
* Secure (program encapsulation) (running machine sandbox limited access to the program)
* Portable and distributable
* Compiled and interpreted
* Robust
* Distributed
* Multithread
* Performance

Quality

Quantity

\*.java 🡪 compile🡪 .class (bytecode)

javac \*.java

run🡪

java \*

\*=filename

# Path

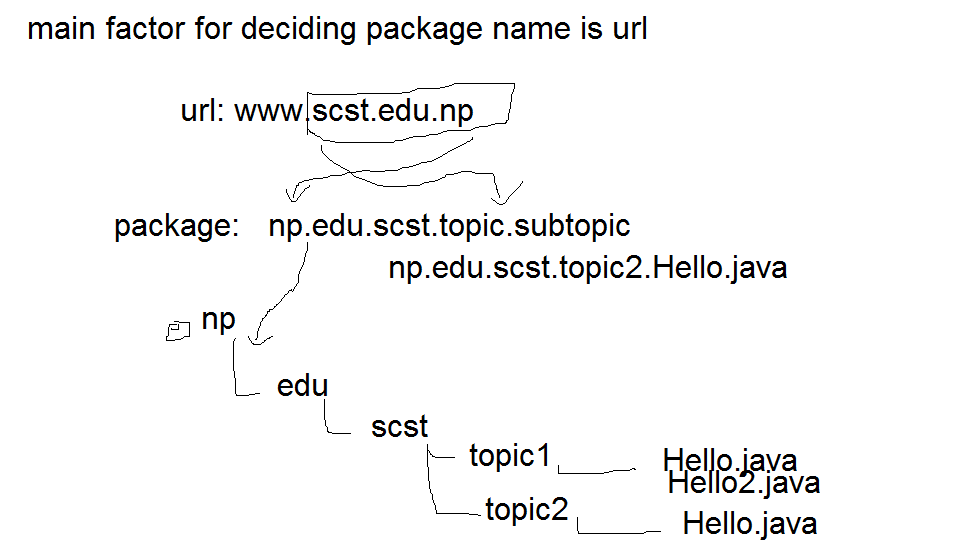
An operating system’s locating mechanism or a locator for identifying the location of executable files.

# Classpath

It is locating mechanism of JVM for finding appropriate classes.

# Package

Package is simply a folder/directory | folder/directory structure, which contains other sub-packages, class files or interface files; for better organization.



# Comment

A text block which is placed inside our program, so as to convey some information. It is readable by us, but is totally ignored by the compiler.

Single line comment: //spans one line

Multi line comment: /\*

Spans multiple lines

\*/

Documentation comment: /\*\*

\* tells information about classes/ interfaces or methods

\*

\*/

# Variable

Variable is a temporary identifier for a data type or object type. It resides inside RAM (Random Access Memory) (main memory).

Rules for naming a variable:

* Variable name must not start with a number. But may contain number after that.
* Variable name must not contain space or any other special characters except ‘\_’.
* Variable name should always represent the meaning they carry.
* Variable name must not collide with that of java reserved names.
* Variable name should be in camelCase.

# Naming conventions:

Package: np.edu.scst.topic.\*\*\*

Variable name: camelCasingFormat

Constant final variable: UPPERCASE

Class/interface: PascalCasing

# Reserved words in java (53):

1. Reserved literals (3): true, false, null
2. Keywords (50)
   1. Unused (2): const, goto
   2. Used (48):
      1. Datatypes 8: byte, short, int, long, float, double, char, boolean (in sequence)
      2. Control statement 10:
         1. Conditionals: if, else, switch, case, default
         2. Loops: for, while, do, break, continue
      3. Method level 2: return, void
      4. Class level 9: package, import, class, interface, abstract, extends, implements, static, enum
      5. Object level 3: this, super, instanceof
      6. Access modifier 5: public, private, protected, volatile, synchronized,
      7. Exception handling 5: try, catch, finally, throw, throws
      8. Thread level 1: transient

# Commonly used variables in java

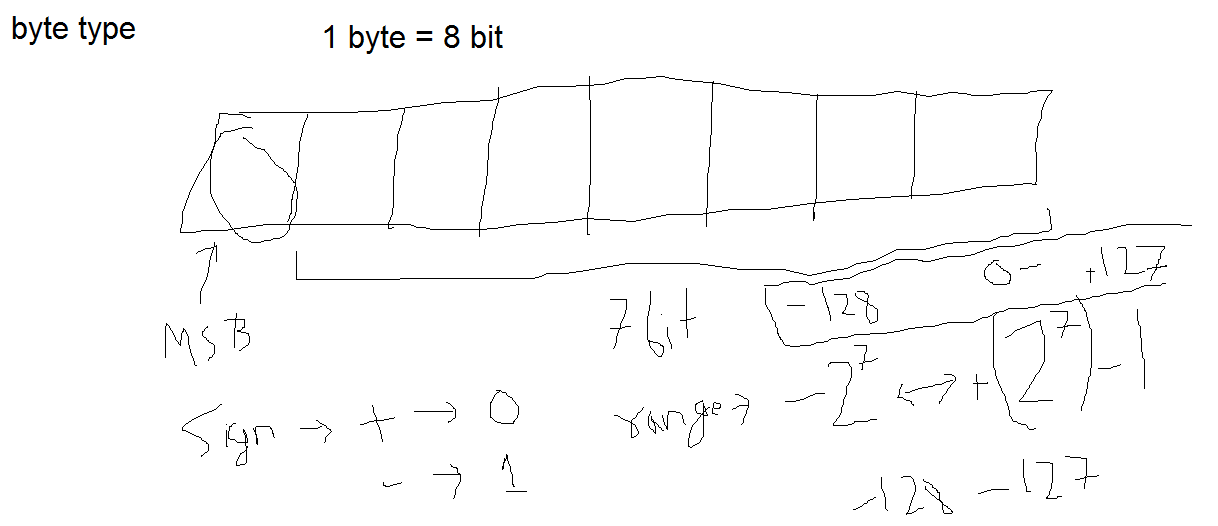
All numerical variable in java are signed.

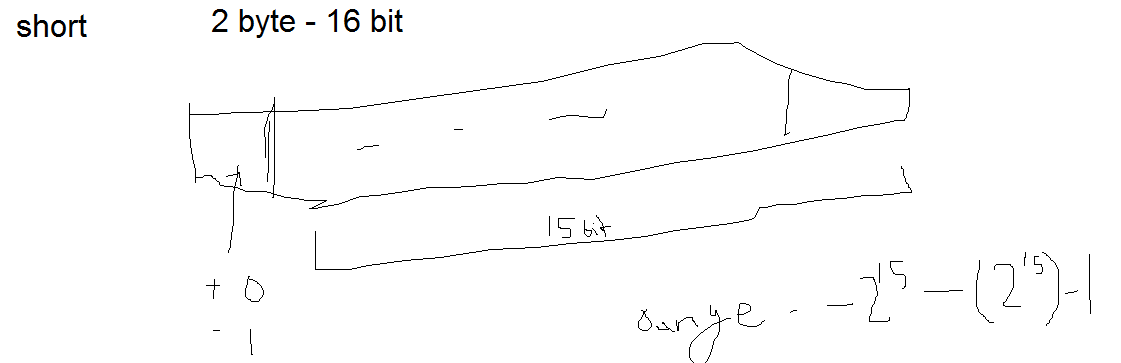
Primitive variables | non-referencing datatypes:

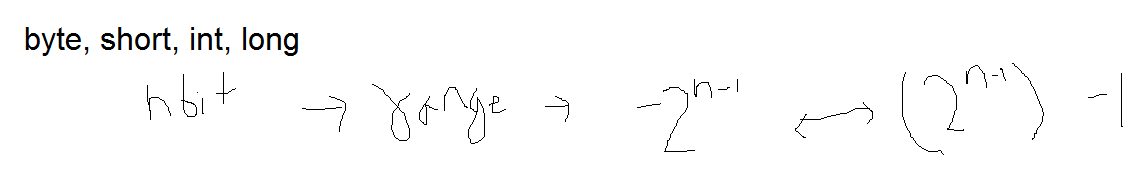
|  |  |  |
| --- | --- | --- |
| byte | 1 byte | -128 - 127 |
| short | 2 byte |  |
| int | 4 byte |  |
| long | 8 byte |  |
| float | 4 byte | 6-7 decimal |
| double | 8 byte | Upto 15 decimal |
| char | 1 byte | Single character ASCII; surrounded by single quotation |
| boolean | 1 bit | true | false |

Java supports Unicode. But java’s char datatype supports ASCII characters.

Java is highly case sensitive.

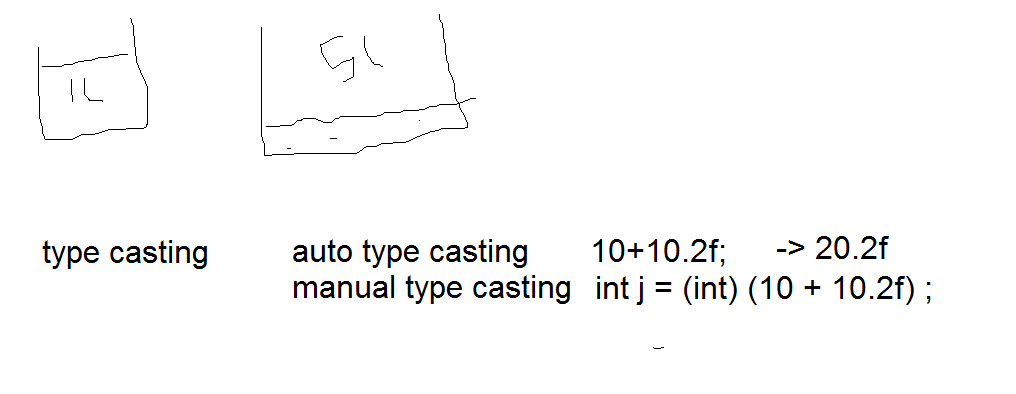






Non primitive datatype | referencing type:

|  |  |
| --- | --- |
| String | Stores sentences. Surrounded by double quotation |

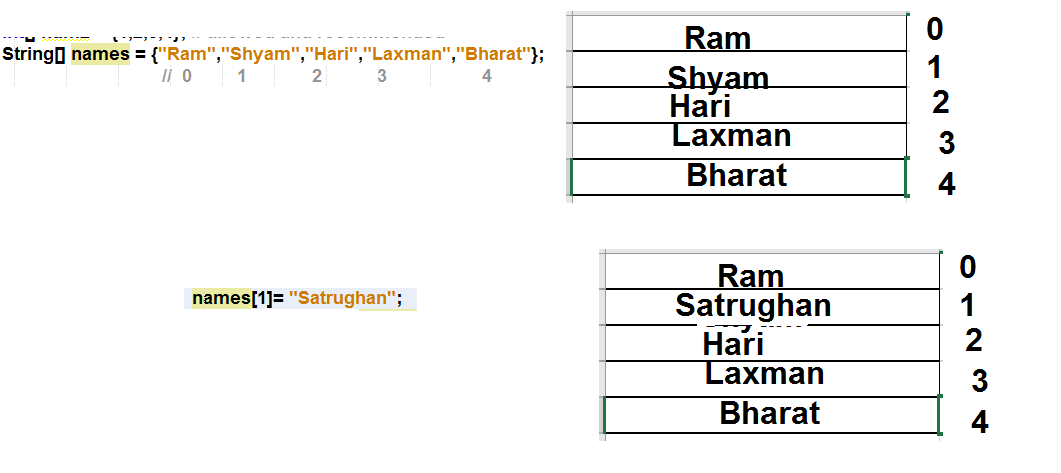


# Array:

Array is an indexed, fixed-sized collection of data elements of same type, stored under a common variable name.

int num1 [] = {1,2,3,4}; //allowed but not recommended

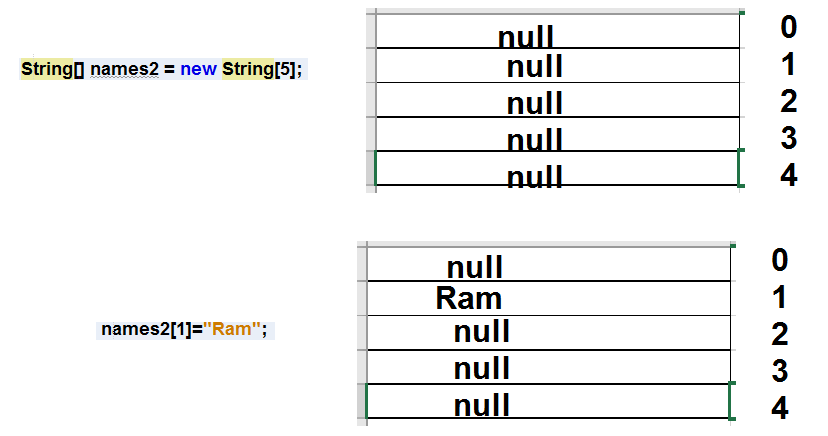
int [] num2 = {1,2,3,4}; // allowed and recommended



String[] names2 = new String[size];

String[] names2 = new String[5];

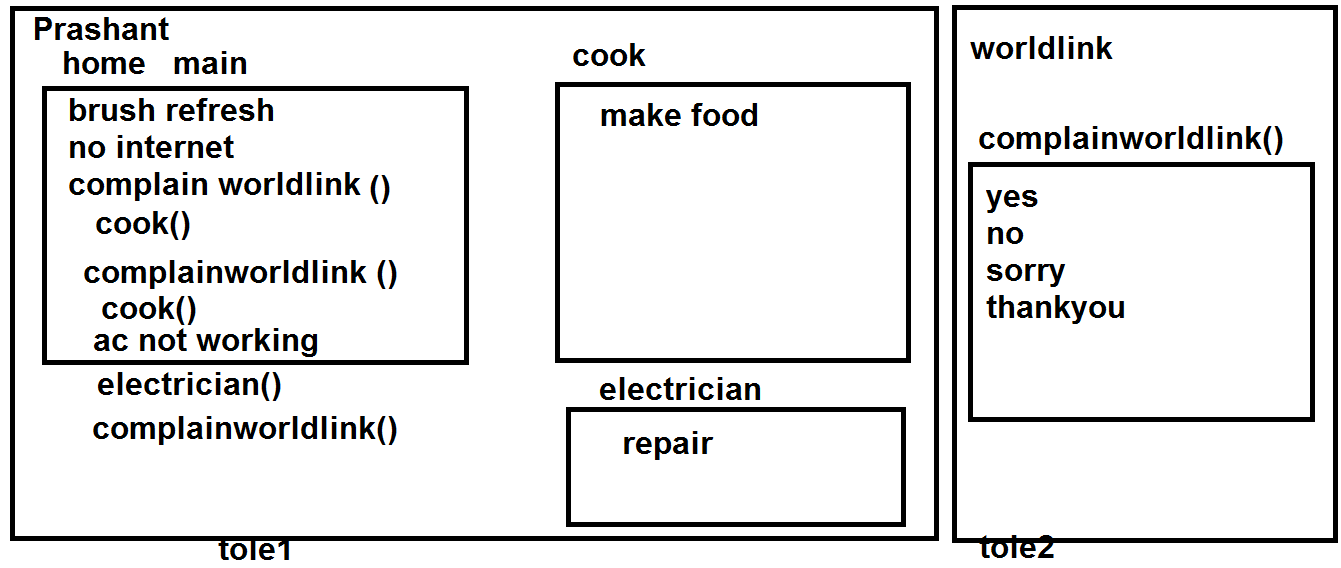
|  |  |  |
| --- | --- | --- |
| Data/classtype | Default value |  |
| int | 0 |  |
| float | 0.0 |  |
| String | null |  |
| boolean | FALSE |  |
| char |  | blank |



# Method vs function:

Function is a re-callable block of code.

Method is a special type of function residing inside a class.



# Classes and objects

Class is a blueprint or template which models real world entity or scenario. Class is a user defined datatype. Class does not occupy memory.

Objects are the physical manifestation of class. Objects are instance of class. Objects are variable of a class. Objects occupy memory.

Class syntax:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| [Access Privilege] | [Modifier] | class | ClassName | extends ParentClass | implements interface1,….. interfaceN |

{

}

Object syntax:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ReferenceClass | objectName | = | new | ClassName(); |
| SelfClass or ParentClass |  |  |  |  |

# Method:

A block of code which can be repeatedly called upon.

Syntax:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Access Privilege | Modifier | returnType | methodName | throws Exception1,…. ExceptonN | (datatype Para1,…. Datatype ParaN ) |

{

}

# Constructor:

A special type of method which is called every time when a new instance (object) of a class is created. A constructor’s name should be same as its corresponding class. A constructor must not have return type.

Syntax:

public ClassName (datatype para1, …. Datatype paraN){

}

Java be default creates an argument less constructor for its classes during compilation.

But when we create any argumented constructor, java will never create a argument less constructor for us. We have to do it manually.

No destructor in java. When scope of an object is completed, JVM’s garbage collector frees up the resources.

Durgasoft youtube

Naveen reddy | telusko

Constructor overloading:

When we have multiple constructors within a class, but with different set of arguments, we call it constructor overloading.

# Inheritance

The process in which child classes (sub class) inherit method and properties from its parent classes (superclass).

|  |  |  |
| --- | --- | --- |
| Single/Simple | Multilevel | Hierarchial |
|  |  |  |
| Muliple | Mixed (multiple+hierarchial) | Not allowed in java |
|  |  |  |

Syntax:

Public class Child extends Parent{

}

# Polymorphism

Overload | compile time polymorphism

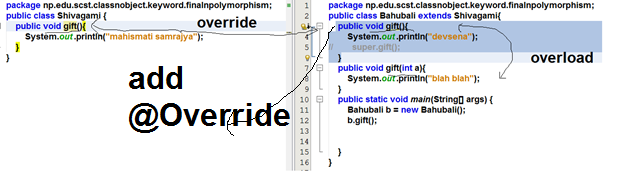
For same class

Override | runtime polymorphism | dynamic method dispatch

For inherited class

dynamic method dispatch

Overloading operator ‘+’



Final keyword:

Access specifier

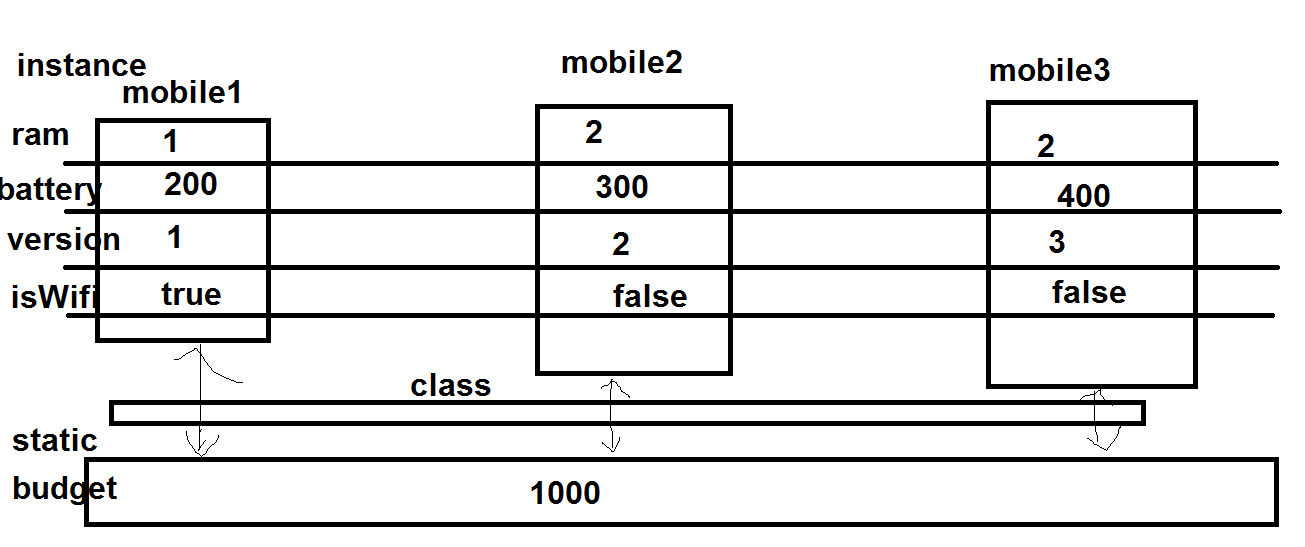
3 uses:

* Variable- when associated with variable, it make that variable constant. i.e. it cannot change after any value is assigned to it
* Method- when attached with any method, the final keyword prevents overriding of the method.
* Class- when attached with any class, its prevents other class from inheriting that particular class; in other words no child of the final class can exist.

Static keyword:

Access specifier:

Denotes class level variables/method; or used to make any variable global. Always call static variable/method through class name.

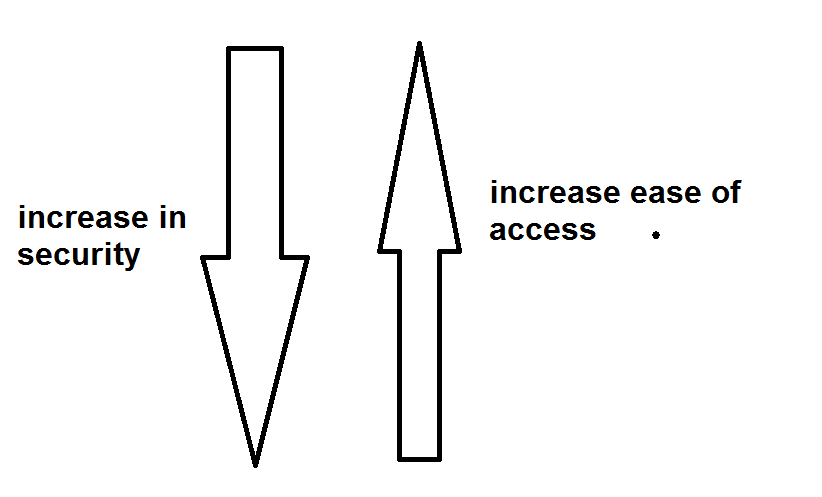


Instance| non-static variables are always unique to a particular object. They don’t have static keyword associated with it. They are called through object name.

# Access privileges / specifiers:

Access privileges / specifier / modifier is used to specify the scope of classes and their properties and methods.

|  |  |  |
| --- | --- | --- |
| Name | Applicable to | Scope |
| public | Class, constructor, variable/property, method | anywhere within the project from anywhere |
| default | Class, variable, method | Within the same package |
| protected | Variable, method | Within the same package + when outside the package, call through child classes |
| private | Variable, method | Within the same class only |



# Abstraction

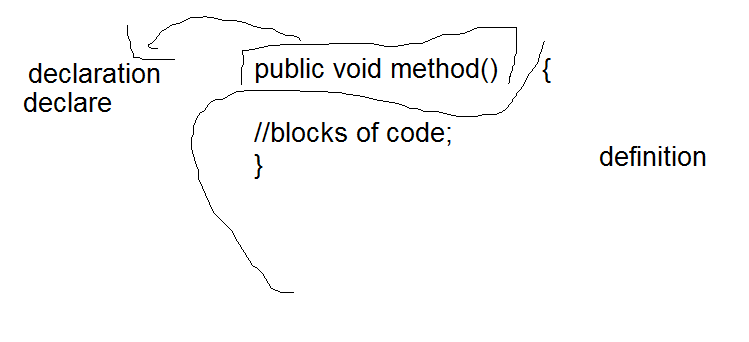
The concept of hiding unnecessary details.

## Abstract class

A special type of class which cannot be instantiated (its object cannot be made) and it may contain abstract methods (special type of method, which contains abstract keyword; we can only declare abstract methods, not define it).

//abstract class may or may not contain abstract method

//but abstract method must always be contained within an abstract class

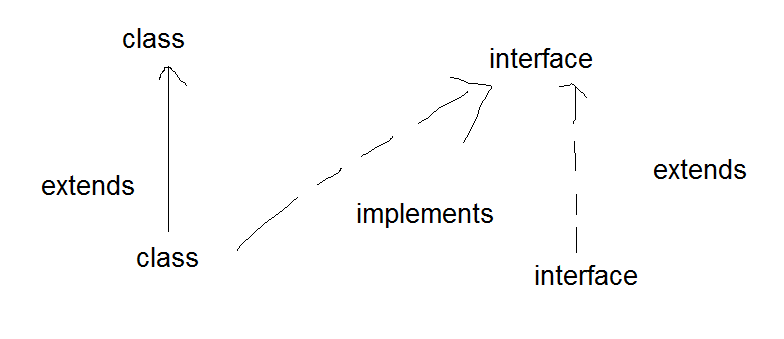


## Interfaces

A special type of abstract class, where all methods are public and abstract by default.

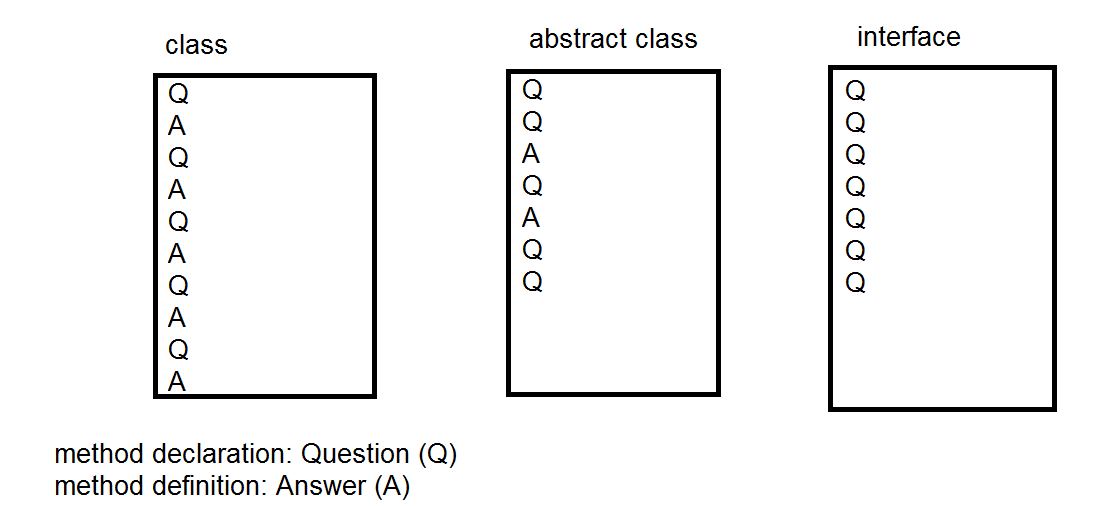
Syntax:

public interface InterfaceName{}



### AbstractClass vs Interface

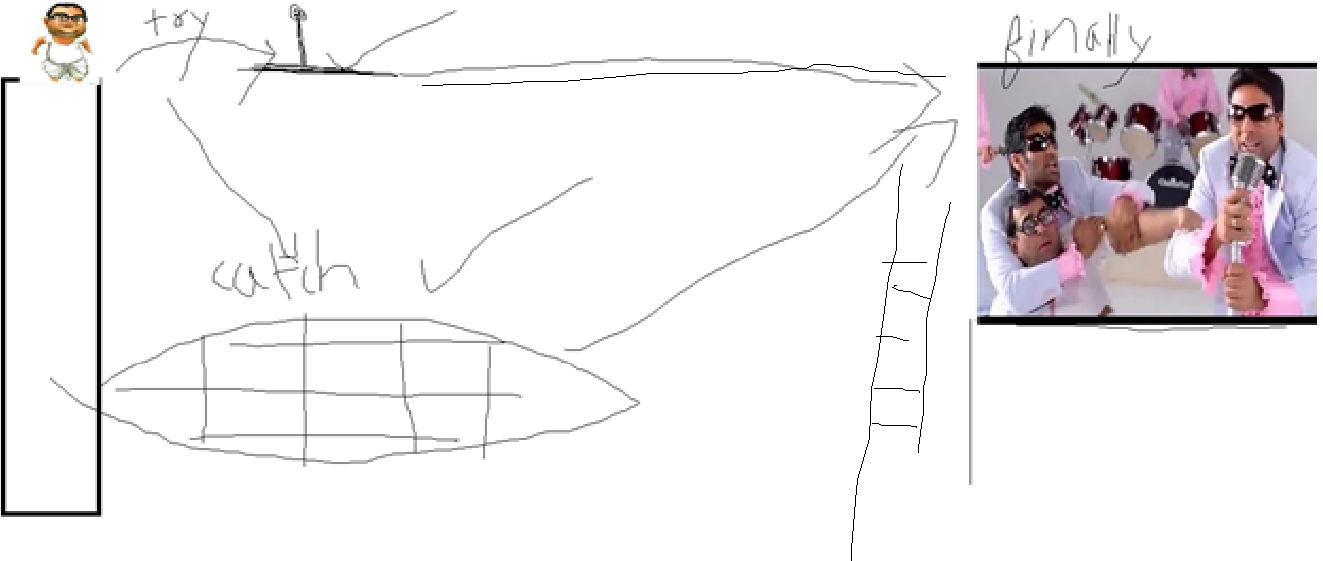
|  |  |
| --- | --- |
| **Abstract class** | **Interface** |
| 1) Abstract class can **have abstract and non-abstract** methods. | Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also. |
| 2) Abstract class **doesn't support multiple inheritance**. | Interface **supports multiple inheritance**. |
| 3) Abstract class **can have final, non-final, static and non-static variables**. | Interface has **only static and final variables**. |
| 4) Abstract class **can provide the implementation of interface**. | Interface **can't provide the implementation of abstract class**. |
| 5) The **abstract keyword** is used to declare abstract class. | The **interface keyword** is used to declare interface. |
| 6) An **abstract class** can extend another Java class and implement multiple Java interfaces. | An **interface** can extend another Java interface only. |
| 7) An **abstract class** can be extended using keyword "extends". | An **interface** can be implemented using keyword "implements". |
| 8) A Java **abstract class** can have class members like private, protected, etc. | Members of a Java interface are public by default. |
| 9)**Example:** public abstract class Shape{ public abstract void draw(); } | **Example:** public interface Drawable{ void draw(); } |



Exception

Exception is an abnormal condition or a problem that arises during a runtime, which results in the abrupt termination of our program (thread).

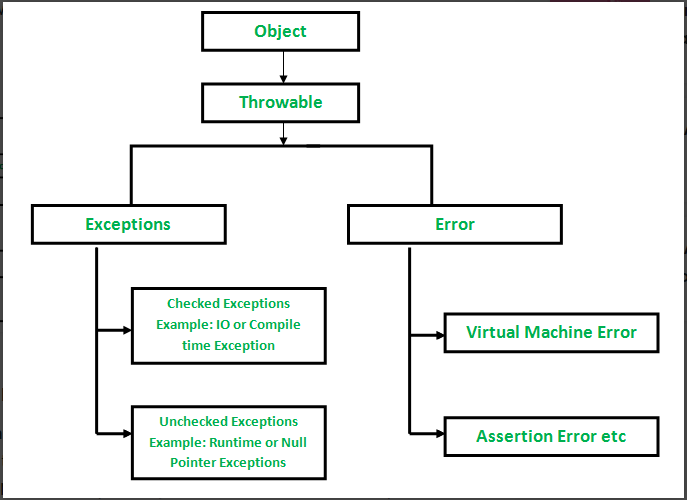




Error: An Error indicates serious problem that a reasonable application should not try to catch.

Errors are problem of system administrator.

Exception: Exception indicates conditions that a reasonable application might try to catch. Exceptions are problem of programmers.



An exception can occur for many different reasons. Following are some scenarios where an exception occurs.

* A user has entered an invalid data.
* A file that needs to be opened cannot be found.
* A network connection has been lost in the middle of communications or the JVM has run out of memory.

**C**hecked exceptions − A checked exception is an exception that is checked (notified) by the compiler at **compilation-time**, these are also called as compile time exceptions. These exceptions cannot simply be ignored; the programmer should take care of (handle) these exceptions. For eg: IOException, SQLException, FileNotFoundException, etc.

Unchecked exceptions − An unchecked exception is an exception that occurs at the time of execution. These are also called as **Runtime Exceptions**. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation. For example, ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException, etc.