JAVA

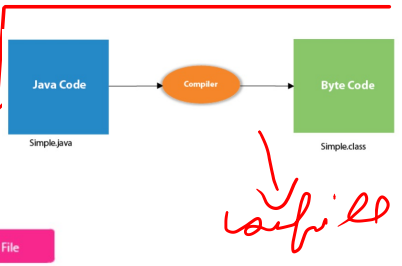
* Programming language
* Platform (Java H/W Environment)

* JAVA APP^N
  - Standalone (JAVA SE)
  - Web (JAVA FX)
  - Enterprise (JAVA EE)
  - Mobile (JAVA ME)

* Features → Simple → OO
  - → Secure → Multithread → Robust
  - → Portable → High Performance
  - → Interpreted → Architecture Independent
  - → Dynamic

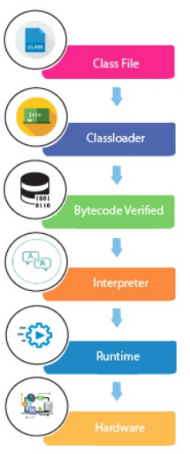
Note: * Java uses its own Runtime env. while other like C++ uses O.S. env.

* Java is secure due to no explicit pointer
* automatic garbage collection



compiled

* Diff b/w JAVA & C++ (https://www.javatpoint.com/cpp-vs-java)
  - → Pointer → Platform Dependency → Security
  - → Virtual → Right Shift → Call by value
  - → Multiple Inheritance → import



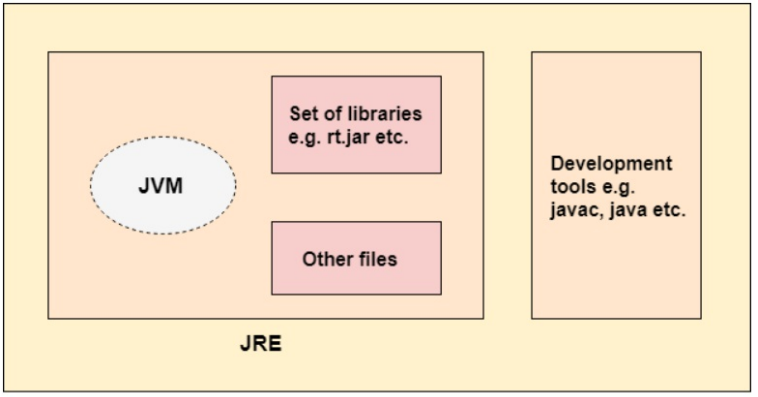
→ Run

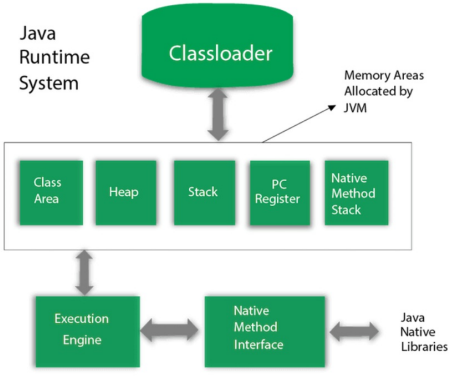
JVM

① Doesn't Exist
② Implemented by JRE

Functions

- Loads code
- Verifies code
- Executes code
- Provides runtime environment





JVM WORKING

1) Classloader

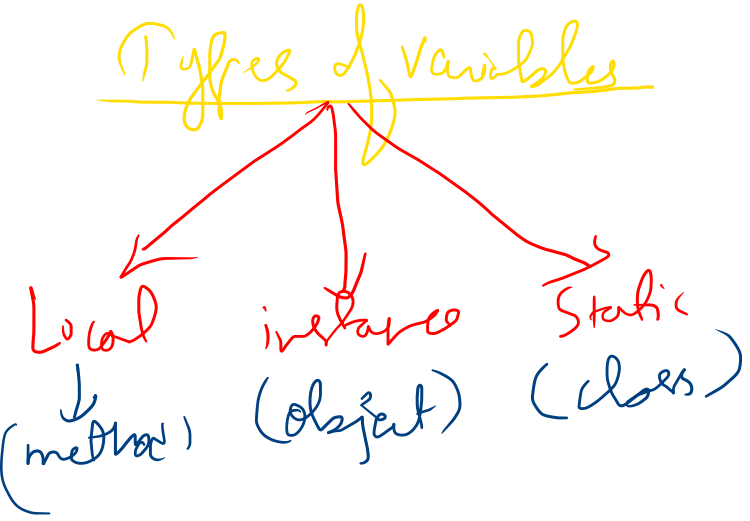
Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

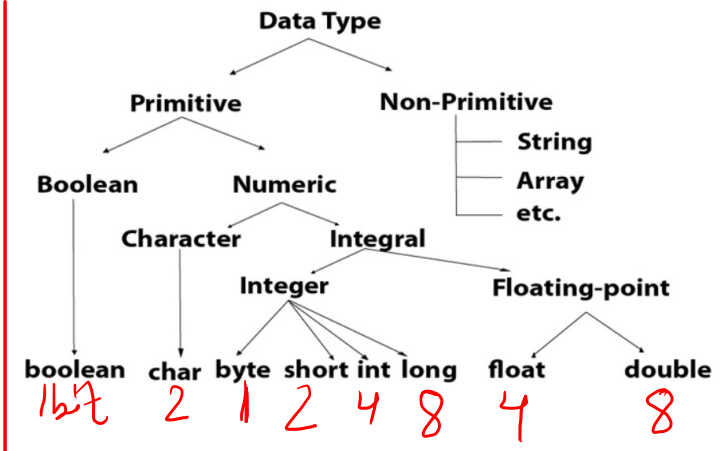
1. **Bootstrap ClassLoader:** This is the first classloader which is the super class of Extension classloader. It loads the *rt.jar* file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.
2. **Extension ClassLoader:** This is the child classloader of Bootstrap and parent classloader of System classloader. It loades the jar files located inside *$JAVA_HOME/jre/lib/ext* directory.
3. **System/Application ClassLoader:** This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

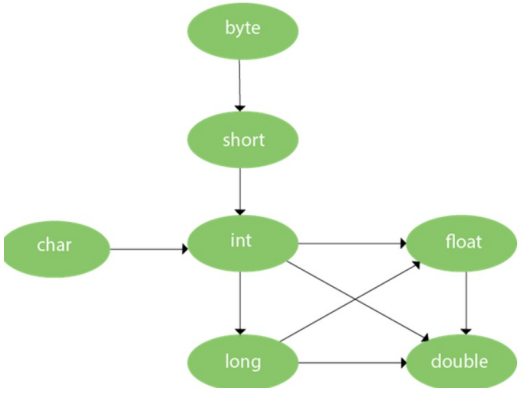
7) Execution Engine

It contains:

1. **A virtual processor**
2. **Interpreter:** Read bytecode stream then execute the instructions.
3. **Just-In-Time(JIT) compiler:** It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.


Types of Variables
Local
(method)
instance
(object)
Static
(class)


Data Type
Primitive
Non-Primitive
String
Array
etc.
Boolean
Numeric
Character
Integral
Integer
Floating-point
boolean
char
byte
short
int
long
float
double
1bit
2
1
2
4
8
4
8


byte
short
char
int
float
long
double

Q) Can you save a java source file by other name than the class name?

↳ Yes, if class is not public

To compile:

javac Hard.java

To execute:

java Simple

Types of Java constructors

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Q) Can you have multiple classes in a java source file?

↳ Yes

```
class A{ }
class B{ }
class C{ }
```

Q) Why multiple inheritance is not supported in java?

To reduce the complexity and simplify the language, multiple inheritance is not supported in java.

What are the different ways to create an object in Java?

There are many ways to create an object in java. They are:

- By new keyword
- By newInstance() method
- By clone() method
- By deserialization
- By factory method etc.

Classes and objects

Class → Blue print for an object

Object → Actual real world entity.

Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.

Polymorphism

If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

Encapsulation

Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Coupling

Coupling refers to the knowledge or information or dependency of another class. It arises when classes are aware of each other. If a class has the details information of another class, there is strong coupling. In Java, we use private, protected, and public modifiers to display the visibility level of a class, method, and field. You can use interfaces for the weaker coupling because there is no concrete implementation.

Cohesion

Cohesion refers to the level of a component which performs a single well-defined task. A single well-defined task is done by a highly cohesive method. The weakly cohesive method will split the task into separate parts. The java.io package is a highly cohesive package because it has I/O related classes and interface. However, the java.util package is a weakly cohesive package because it has unrelated classes and interfaces.

Association

Association represents the relationship between the objects. Here, one object can be associated with one object or mar objects. There can be four types of association between the objects:

Aggregation

Aggregation is a way to achieve Association. Aggregation represents the relationship where one object contains other objects as a part of its state. It represents the weak relationship between objects. It is also termed as a *has-a* relationship in Java. Like, inheritance represents the *is-a* relationship. It is another way to reuse objects.

Composition

The composition is also a way to achieve Association. The composition represents the relationship where one object contains other objects as a part of its state. There is a strong relationship between the containing object and the dependent object. It is the state where containing objects do not have an independent existence. If you delete the parent object, all the child objects will be deleted automatically.

OOPS → Java, Procedural → C/C++, Functional → Javascript

STATIC KEYWORD

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block
4. Nested class

↳ static means related to class
↳ used for memory management
↳ static can call static only

THIS KEYWORD

Usage of java this keyword

Here is given the 6 usage of java this keyword.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() can be used to invoke current class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method.

↳ this refers to current object
↳ differentiate from local and instance variable

SUPER KEYWORD

- super variable refers immediate parent class instance.
- super variable can invoke immediate parent class method.
- super() acts as immediate parent class constructor and should be first line in child class constructor.

↳ next parent of current class.

FINAL, FINALLY & FINALIZE

1. **Final** → Bound extension of something.
   - ↳ Variable → Variable become constant.
   - ↳ method → that method cannot be overrides.
   - ↳ class → that class cannot be extended by others.

2. **Finally**
   - ↳ A part of Try catch which always execute

3. **finalize**
   - ↳ method called by garbage collector

Note: Final method is inherited but can't override.

★ Uninitialized final → only initialized in constructor (Before object)

★ Static final → only initialized in static block (Before program)

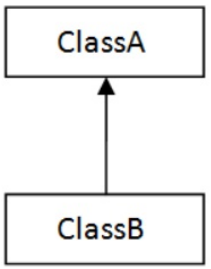
Q) Can we declare a constructor final?

No, because constructor is never inherited.

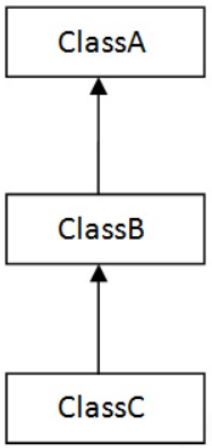
A signature can include:

- parameters and their types
- a return value and type
- exceptions that might be thrown or passed back
- information about the availability of the method in an object-oriented program (such as the keywords `public`, `static`, or `prototype`).

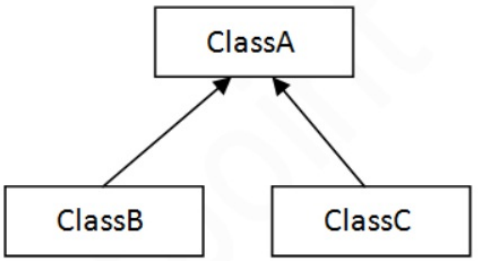
TYPES OF INHERITANCE IN JAVA



1) Single



2) Multilevel



3) Hierarchical

* for code reusability
* for polymorphism

Types of relationships

IS-A

(extends) INHERITANCE

HAS-A

ASSOCIATION (Object)

AGGREGATION (Weak)

COMPOSITION (Strong)

POLYMORPHISM

↳ Many behaviour.

1. Method Overloading (Static binding)
   - ↳ within same class
   - ↳ Compile time resolution

2. Methodoverriding (Dynamic Binding)
   - ↳ method of other class / interface is overridden.
   - ↳ Run time resolution

RUNTIME POLYMORPHISM

↳ First create object ↳ from ref. check for overriding.
↳ move from top to down. (last overrider)

Note: if we create object for subclass then constructor for superclass are called automatically. and here we have data for both classes.

P{}

C extends P{}

```
P obj2 = new P();
C obj = new C();
P obj = new C();
C obj = new P();
```

(obj = new P(); — wrong bcos P doesn't have C's data.)

Data members are not overridden & hence depends of reference (LHS).

ABSTRACT CLASS

1. Can have abstract method
2. Can have non abstract method
3. Can have instance variables.
4. Cannot be instantiated (No idea "How Much Memory")
5. A class having abstract method must be abstract
6. A class extending abstract class should either be abstract or implement all abstract methods.

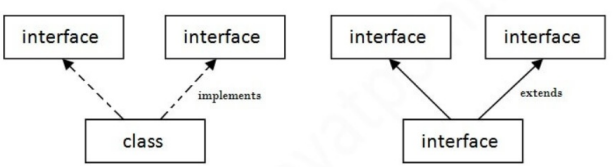
Note: Abstract method can have constructor to initialize non abstract or instance variables

Note: Constructor of class extending abstract class calls constructor of abstract class by default

Uses
↳ Don't provide complete implementation.

INTERFACE IN JAVA

1. abstract method or default methods.
2. Data members are Public, Static, Final.
3. cannot be instantiated
4. uses implements with class & extend for interface
5. multiple inheritance can be achieved.
6. 2 interface can't have same signature of any method if implemented simultaneously.
7. cannot have constructor
8. can have private methods (for nested interfaces)



Multiple Inheritance in Java

Diff b/w Abstract class & Interfaces

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();

PACKAGE IN JAVA

A **java package** is a group of similar types of classes, interfaces and sub-packages.

Package in java can be categorized in two form, built-in package and user-defined package.

There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

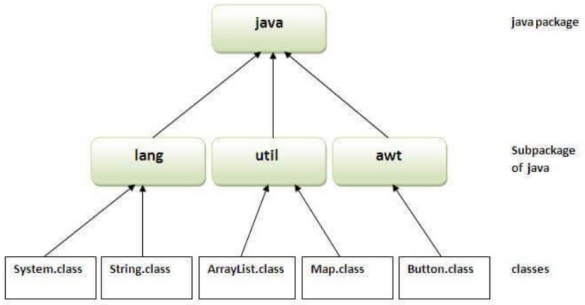
Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.

2) Java package provides access protection.

3) Java package removes naming collision.



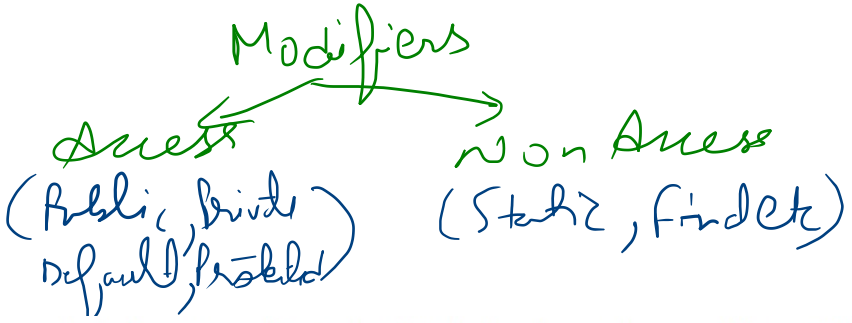
1. Compiling

```
javac -d directory javafilename
```

2. Running : java mypack.Simple

3. Attaching

1. import package.*;
2. import package.classname;
3. fully qualified name.



1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

ENCAPSULATION (use of Private)

Encapsulation in Java is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.

The **Java Bean** class is the example of a fully encapsulated class.

Advantage of Encapsulation in Java

By providing only a setter or getter method, you can make the class **read-only or write-only**. In other words, you can skip the getter or setter methods.

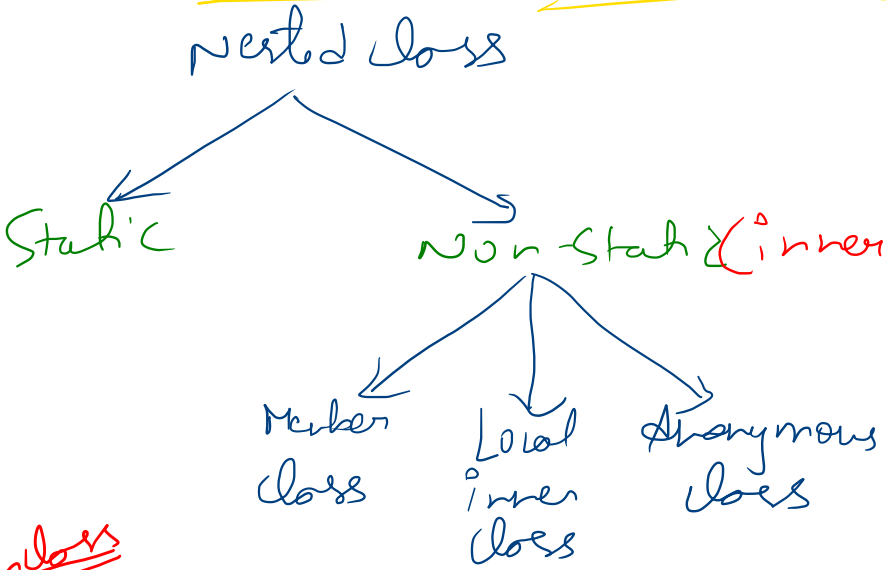
It provides you the **control over the data**. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.

It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.

The encapsulate class is **easy to test**. So, it is better for unit testing.

The standard IDE's are providing the facility to generate the getters and setters. So, it is **easy and fast to create an encapsulated class** in Java.

NESTED CLASSES IN JAVA



→ Non static classes are called inner classes b'cos they are part of objects.
→ Readable & maintainable
→ Code optimization

Inner class

Member Inner Class	A class created within class and outside method.
Anonymous Inner Class	A class created for implementing interface or extending class. Its name is decided by the java compiler.
Local Inner Class	A class created within method.
Static Nested Class	A static class created within class.
Nested Interface	An interface created within class or interface.

Static class

* Linked with outer class directly.
* Cannot call non static part of outer class
* can access private members of static.
* object → new A.B();

Non static class

Member Inner class (non-static + outside method)

- Object → `A.B obj = (new A()).new B();`
- Can call static / non static / Private of outer class.

Local Inner class (non static + inside method)

- only be instantiated in method.
- cannot be public.
- can access all outer methods or classes part.

Anonymous Inner class

- doesn't have any name. (name given by JVM).
- used to override method of class / interface

use → `Parent p = new Parent() { @override method };`

Direct overriding method of Parent class.

EXCEPTION HANDLING

Ex → Daily routine + bike puncture.

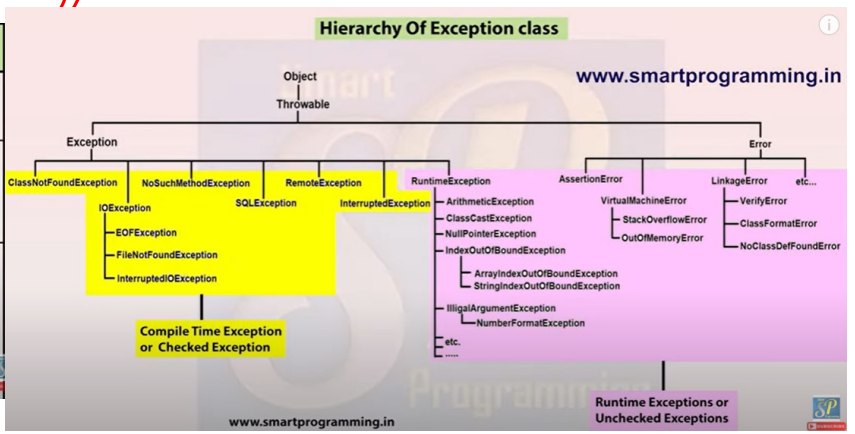
Exception → 0 / Critical Statement behaves (-ve)

Exception Handling → Alternative way to proceed if exception occurs.

Note: Object class is parent class of all classes.

↳ Throwable is parent of exception class.

Exception	Error
1. Exception occurs because of our programs	1. Error occurs because of lack of system resources.
2. Exceptions are recoverable i.e. programmer can handle them using try-catch block	2. Errors are not recoverable i.e. programmer can handle them to their level
3. Exceptions are of two types : ▪ Compile Time Exceptions or Checked Exceptions ▪ Runtime Exceptions or Unchecked Exceptions	3. Errors are only of one type : ▪ Runtime Exceptions or Unchecked Exceptions



Note: No exception occurs at compile time.

↳ At compile time we get a warning of future exceptions.

Compile Time Vs Runtime Exception

↳ which can be checked by compiler i.e. prediction of exception by compiler.

Ex ⇒ FileInputStream exception for FileNotFound.

↳ which can not be checked by compiler.

Ex ⇒ Dividing by '0'.
↳ null pointer.

Process for Exception Handling

(5-1) exception object creation (5-2) Send to JVM

(5-3) if Handled not → terminate and print object.

TRY-CATCH-FINALLY

① Try (Find Exception)

- ☆ Contains risky code ☆ break "try" as soon as exception found.
- ☆ must follow ☆ catch / finally ☆ put risk code in try.
- ☆ if no exception is found in try then catch doesn't execute

② Catch (Handle Exception)

☆ only after catch block ☆ can be multiple for single "try"

1. e.printStackTrace() → Print everything
2. e / e.toString() → Exception Stack Trace
3. e.getMessage() → only Description

Exp. name
Description
Stack trace

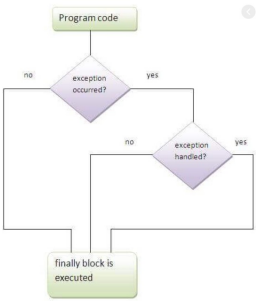
③ Finally

☆ always execute ☆ must be with try ☆ can be single

☆ use → closing resources

Note: If we return from try then also finally execute

↳ System.exit(0) doesn't execute finally or due to errors or Exception in finally



THROW KEYWORD (here throw class()),

Q if an exception occurs then method can create object then why to use throw to make object

↳ To Handle user-defined exception. Ex - Voter with age < 18 should not vote in voting clg.

Ex: class YoungAgeException extends RuntimeException{

YoungAgeException(String msg){

super(msg);

}

psvm(String->){

if(age<18){ throw new YoungAgeException("msg");

}

THROWS KEYWORD (throws class)

↳ If we don't want to handle exception in current method then we throws an exception and order the calle method to handle the exception

↳ It's good practise to use try catch in main() then using throws b'cos throws in main() results in termination of program.

Note: used only for checked exception.

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws is used with the method signature.
5)	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.