Core Java

Devarajan S
Technical Trainer
email:vnrajakademy@gmail.com

Introduction to Java

- **Developed by Sun Microsystems in 1991 later acquired by Oracle**
- **Conceived by James Gosling and Patrick Naughton**
- **Runs on variety of platforms – Windows, Mac OS and UNIX**
- **Multiple configurations were built to suite various types of platorms - J2EE for Enterprise Applications, J2ME for Mobile Applications**
- **Java is guaranteed to be Write Once, Run Anywhere**

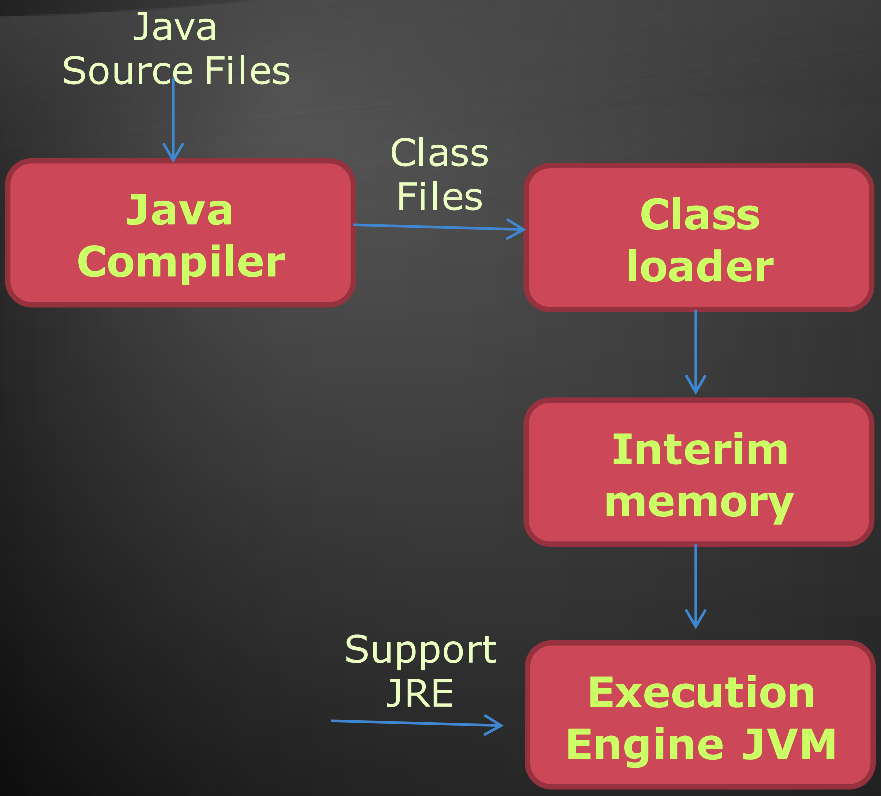
Main Features of Java

- **Platform Independent**
- **Object Oriented**
- **Simple – Most of the concepts are drew from C++**
- **Secure**
  - Provides a secure means of creating and accessing web applications
  - Java codes are confined within Java Runtime Environment (JRE) thus it does not grant unauthorized access on the system resources
- **Distributed**
  - RMI and EJB are used for creating enterprise applications
  - The java programs can be distributed on more than one systems that are connected to each other using internet connection
- **Multithreading**
- **Architecture-Neutral**
- **Portable**
- **High Performance**

Architecture Neutral & Portable

- Java Compiler -Java source code to *bytecode*
- *Bytecode* - an intermediate form, closer to machine representation
- A virtual machine on any target platform interprets the bytecode
- Porting the java system to any new platform involves writing an interpreter that supports the Java Virtual Machine
- The interpreter will figure out what the equivalent machine dependent code to run

Architecture of Java



- **Internal Memory consists of heap, stacks and registers to store data**
- **JRE has native methods and libraries**
- **JVM runs two threads:**
  - **Demon – Garbage Collector**
  - **Non-demon – Main()**

JDK vs JRE vs JVM

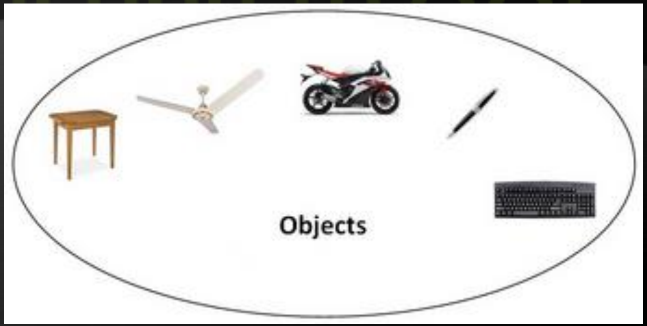
- JDK
  - Physical entity – Contains JRE and development tools
- JRE
  - Physical entity – Implementation of JVM
  - Contains set of libraries and other files that JVxM uses at runtime
- JVM
  - Abstract machine
  - Tasks: Load, Verifies and Executes Code

- JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent

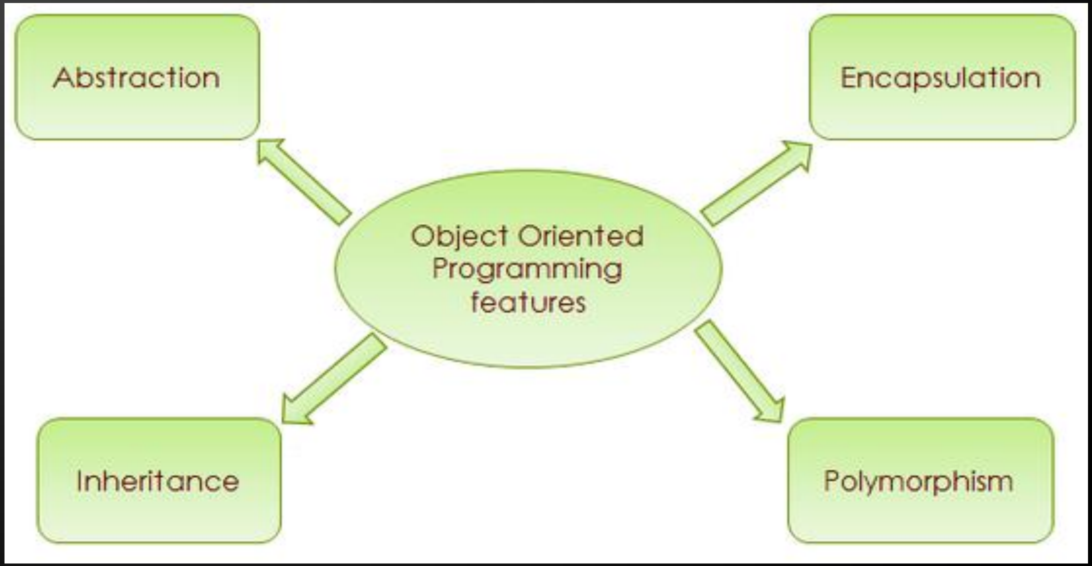
Introduction to OOP (contd…)

- **Class**
  - A template/blueprint that describes the behaviour/states common to all objects of a certain kind
  - Member functions operate upon the member variables of the class
  - A class is a Type similar to int and boolean – E.g. Student std;
  - When a class is instantiated:
    - Object is created– Eg. Std = new Student();
    - Memory space is allocated
  - Many classes contain only static members, or only non-static. However, it is possible to mix static and non-static members in a single class
  - Contain following variable types:
    - Local Variables
    - Instance Variables (Non-static)
    - Class Variables (Static)
    - **Defining a class variable :**
      E.g. **GraduationCourse mycourse= new GraduationCourse();**

Introduction to OOP



- **Object**
  - **Bundle of Data + Instructions**
  - **Share two characteristics**
    - **State – Captures the relevant aspects of an object**
    - **Behaviour – Observable effects of an operation or event**
  - **Instance of a class**
- **Advantages**
  - Objects are modeled on real world entities
  - enables modeling complex systems of real world into manageable software solutions



Introduction to OOP (contd…)

- **Characteristics of Objects:**
  1. Abstraction
  2. Encapsulation
  3. Message passing

- **Abstraction**

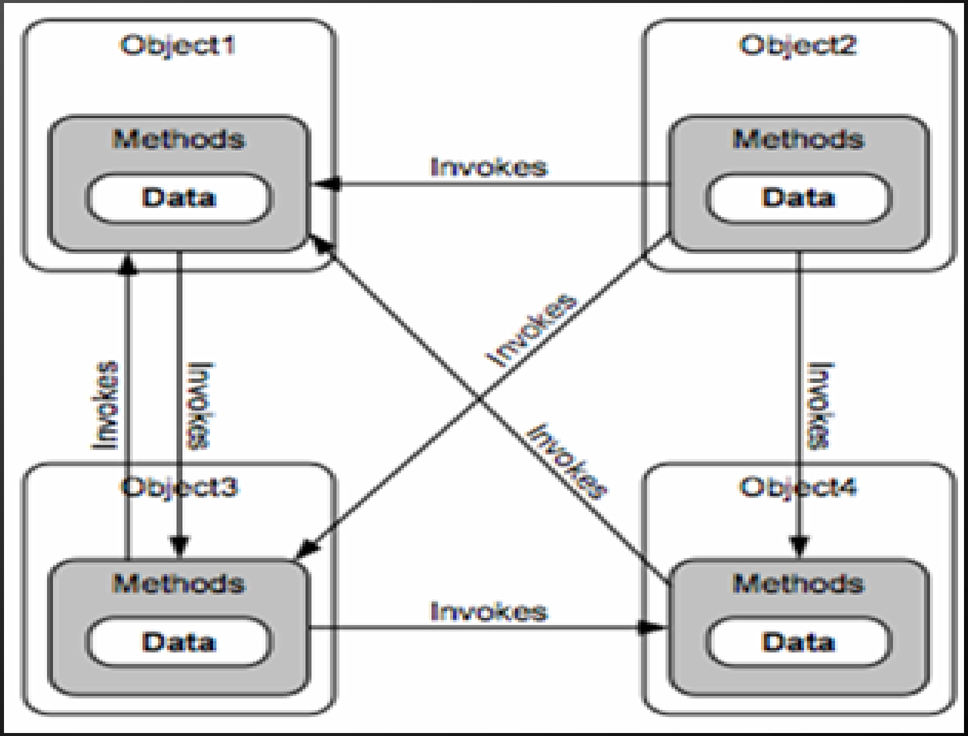
  "The process of forming of general and relevant concept from more complex scenarios"

  - Show only relevant information
  - Hiding internal details and showing functionality
  - In java, we use abstract class and interface to achieve abstraction

- **Encapsulation**
  - Binding (or wrapping) code and data together into a single unit is known as encapsulation
  - To the outside world, an object is a black box that exhibits a certain behavior
  - The set of functions an object exposes to other objects or external world acts as the interface of the object

Introduction to OOP (contd…)

- **Message Passing (Method Invocation)**
  - Through the interaction of objects, programmers achieve a higher order of functionality



Introduction to OOP (contd…)

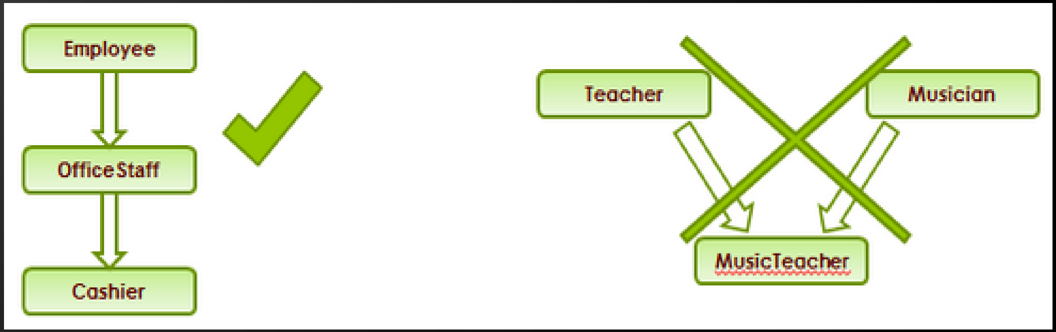
- **How to bring in Encapsulation**
  - Make the instance variables protected
  - Create public accessor methods and use these methods from within the calling code
  - Use the naming convention of **getter and setter**
    Eg: getPropertyName, setPropertyName.

```
class Encapsulation
{
   public static void main(String args[])
   {
      System.out.println("Starting EmployeeCount...");
      EmployeeCount employeeCount = new EmployeeCount (); //Another class object
      employeeCount. setNoOfEmployees (12003);
      System.out.println("NoOfEmployees = " + employeeCount. getNoOfEmployees ());
    }
}
```

Introduction to OOP (contd…)

- **Inheritance**
  - **One object acquires all the properties and behaviours of parent object**
  - **Code reusability**
  - **Used to achieve runtime polymorphism**
  - **Subclass cannot access members of the super class that are declared as private**
  - **A reference variable of a super class can be assigned to a reference to any sub class derived from that super class**
    i.e. Employee emp = new Teacher();

```
class Super{

.....

}


class Sub extends Super{

.....

}
```



Multi-Level Inheritance

Introduction to OOP (contd…)

- **Polymorphism**
  - an operation may exhibit different behavior in different instances
  - extensively used in implementing inheritance
  - Types of Polymorphism
    1) Static Polymorphism - Function Overloading
    2) Dynamic Polymorphism – Function Overriding
  - **Overriding**:
  - Redefining a super class method in a sub class is called method overriding
  - Method signature ie. method name, parameter list and return type have to match exactly
  - Overridden method can widen the accessibility but not narrow it, i.e. if it is private in the base class, the child class can make it public but not vice versa

```
Doctor doctorObj = new Doctor();
doctorObj.treatPatient()
Surgeon surgeonObj = new Surgeon();
surgeonObj.treatPatient()
Doctor obj = new Surgeon();
obj.treatPatient(); // calls Surgeon’s treatPatient method as the reference is pointing to Surgeon
```

Introduction to OOP (contd…)

- **Abstract Class**
  - Outlines the behavior but not necessarily implements all of its behavior. Also known as **Abstract Base Class**
  - Provides outline for behavior by means of method (abstract methods) methods) **signatures** without an implementation
  - A class derived from the abstract base class must implement those member functions which are not implemented in the abstract class else subclass will also become abstract
  - Abstract Class cannot be instantiated

E.g.

```
abstract class Costume {
  abstract public void Stitch();
   public void  setColour (){
     //code to set colour
   }
}
```

Introduction to OOP (contd…)

- **Abstract Class**
  - Abstract classes may or may not contain abstract methods
  - if a class have at least one abstract method, then the class must be declared abstract
- Following cannot be marked with “abstract” modifier
  - Constructors
  - Static methods
  - Private methods
  - Methods marked with “final” modifier

Introduction to OOP (contd…)

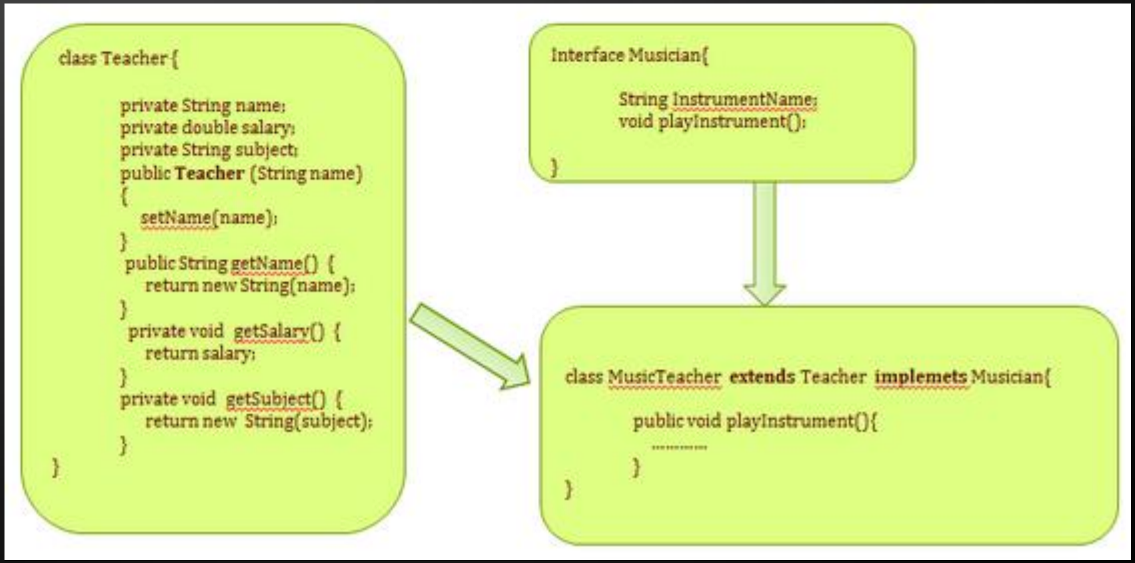
Constructor:

- A method with the same name as class name used for the purpose of creating an object in a valid state
- It does not return a value not even void
- It may or may not have parameters (arguments)
- **All objects are created through Constructors**
- **The default constructor sets instance variables as:**
    - numeric types are set to zero
    - boolean variables are set to false
    - char variables are set to ‘’
    - object variables are set to null

Interface

- An interface can be defined as a contract between objects on how to communicate with each other
- Similar to an abstract class that contains only abstract methods
- An interface defines the methods a subclass should use. But the implementation of the methods is totally up to the subclass
- Keyword implements is used to represent the interfaces implemented by the class

class ClassName extends Superclass implements Interface1, Interface2, ...



```
class Teacher {
        private String name;
        private double salary;
        private String subject;
        public Teacher (String name)
        {
           setName(name);
        }
        public String getName() {
           return new String(name);
        }
        private void getSalary() {
           return salary;
        }
        private void getSubject() {
           return new String(subject);
        }
}
```

```
Interface Musician{
        String InstrumentName;
        void playInstrument();
}
```

```
class MusicTeacher extends Teacher implemets Musician{
        public void playInstrument(){
           ...........
        }
}
```

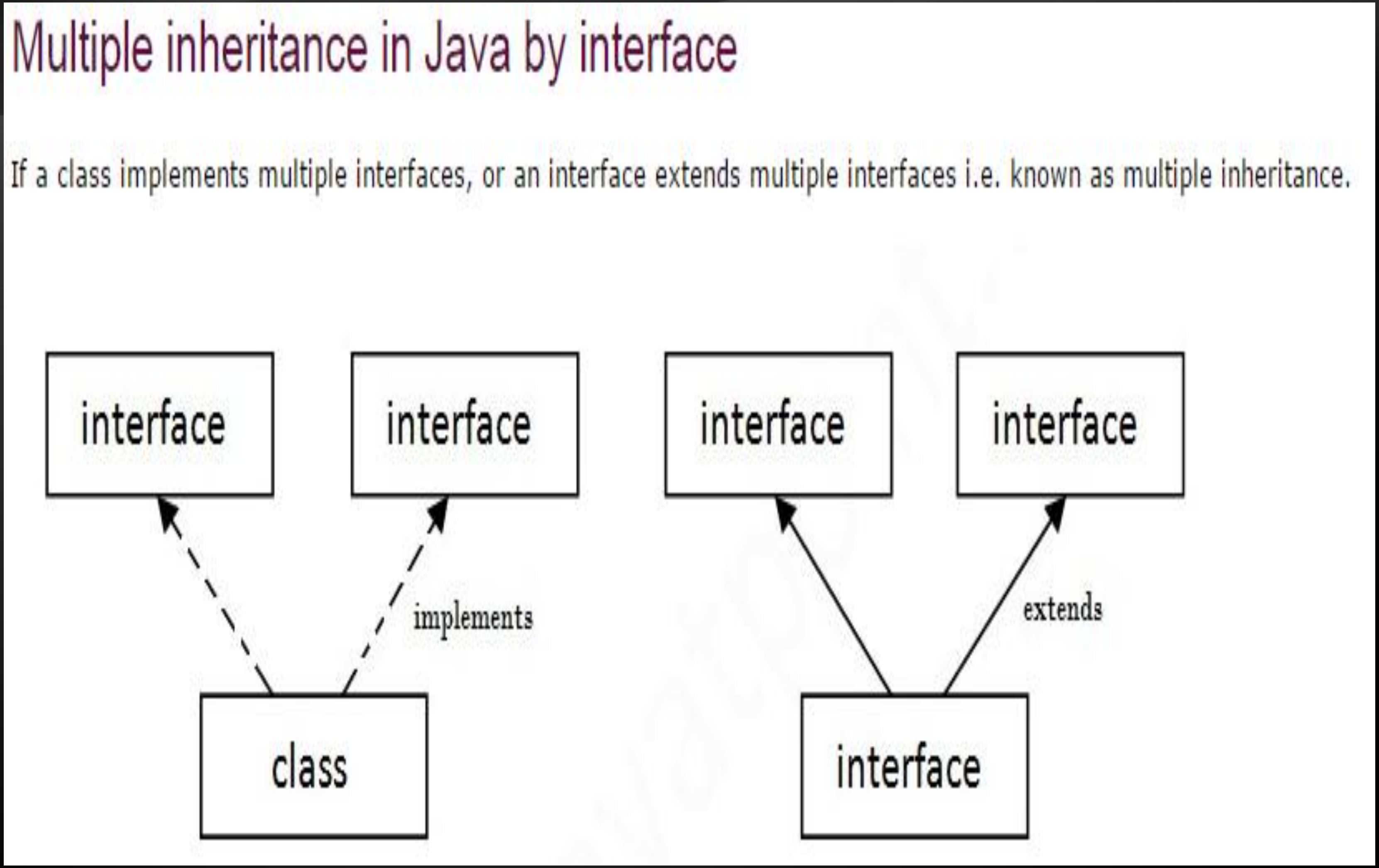
Interface

- *can contain any number of methods*
- *All **methods in an interface** are implicitly public and abstract*
- *A class may **extend only one other class**, but it may **implement any number of interfaces***
- *When a class implements an interface, it may implement (define) some or all of the **inherited abstract methods***

Interface

- *An interface is written in a file with a .java extension, with the name of the interface matching the name of the file*
- *The byte code of an interface appears in a .class file.*
- *Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.*
- *However, an interface is different from a class in several ways, including:*
- *You cannot instantiate an interface.*
- *An interface does not contain any constructors.*
- *All of the methods in an interface are abstract.*
- *An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.*
- *An interface is not extended by a class; it is implemented by a class.*
- *An interface can extend multiple interfaces.*

Interface


Multiple inheritance in Java by interface

If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance.

Access Modifiers

- **public:** Accessible to all. Other objects can also access this member variable or function
- **private:** Not accessible by other objects. Private members can be accessed only by the methods in the same class. **Object accessible only in Class in which they are declared**
- **protected:** The scope of a protected variable is within the class which declares it and in the class which inherits from the class (Scope is Class and subclass)
- **Default:** Scope is Package Level

- There are other Modifiers in Java. They are
  - static
  - abstract
  - final
  - synchronized
  - transient
  - native
  - volatile

Access Modifiers

- **static:**
  - **Mainly used for memory management**
  - **Can be used with variables, methods, blocks and nested class**
  - **belongs to the class than instance of the class**
- **Static Variables**
  - **The static variable can be used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.**
  - **The static variable gets memory only once in class area at the time of class loading (memory efficient as it saves memory)**
  - **Java static property is shared to all objects**

    ```
    static int count=0;//will get memory only once and retain its value
    ```
  - if any object changes the value of the static variable, it will retain its value

Access Modifiers

- **Static method:**
  - belongs to the class rather than object of a class
  - can be invoked without the need for creating an instance of a class
  - can access static data member and can change the value of it

- Restrictions for static method
  - There are two main restrictions for the static method. They are:
    - The static method can not use non static data member or call non-static method directly
    - this and super cannot be used in static context

```
class A{
 int a=40;//non static

 public static void main(String args[]){
  System.out.println(a);
 }
}
```

Output: Compile Time Error

Final Keyword

- **Final variable:**
  - Final variables are nothing but constants.
  - We cannot change the value of a final variable once it is initialized
  - If not initialized, it must be initialized in the constructor

```
class Demo{

  final int MAX_VALUE=99;
  void myMethod(){
    MAX_VALUE=101;
  }
  public static void main(String args[]){
    Demo obj=new  Demo();
    obj.myMethod();
  }
}
```

Output:

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
    The final field Demo.MAX_VALUE cannot be assigned
```

Final Keyword

- **Final method:**
  - Cannot be overridden
  - Though a sub class can call the final method of parent class without any issues but it cannot override it

```
class XYZ{
   final void demo(){
      System.out.println("XYZ Class Method");
   }
}

class ABC extends XYZ{
   public static void main(String args[]){
      ABC obj= new ABC();
      obj.demo();
   }
}
```

Final Keyword

- **Final method:**
  - Cannot be overridden
  - Though a sub class can call the final method of parent class without any issues but it cannot override it

```
class XYZ{
   final void demo(){
      System.out.println("XYZ Class Method");
   }
}

class ABC extends XYZ{
   public static void main(String args[]){
      ABC obj= new ABC();
      obj.demo();
   }
}
```

Final Keyword

1) A **constructor** cannot be declared as final.
2) Local final variable must be initializing during declaration.
3) All variables declared in an **interface** are by default final.
4) We cannot change the value of a final variable.
5) A final method cannot be overridden.
6) A final class not be inherited.
7) If method parameters are declared final then the value of these parameters cannot be changed.
8) It is a good practice to name final variable in all CAPS.

Approach to Object Oriented Design

- Step 1. Identify all the classes (Nouns; Type of objects) in the requirement specification,
- Step 2. Identify the commonalities between all or small groups of classes identified (generalization) if it is obvious. Do not force fit generalization where it doesn't make sense.
- Step 3. In any given situation, start with the simplest object which can be abstracted into individual classes
- Step 4. Identify all the member variables and methods the class should have
- Step 5. Ensure that the class is fully independent of other classes and contains all the attributes and methods necessary.
- Step 6. Keep all the data members private or protected
- Step 7. The methods in the class should completely abstract the functionality
- Step 8. The methods in the class should not be a force fit of procedural code into a class
- Step 9. Inherit and extend classes from the base classes ONLY IF the situation has scope for it
- Step 10. Define the relationships among the classes ("Has-A", "Uses-A")
- Step 11. Keep the number of classes in your application under check (do not create any unnecessary classes)its from the class (Scope is Class and subclass)

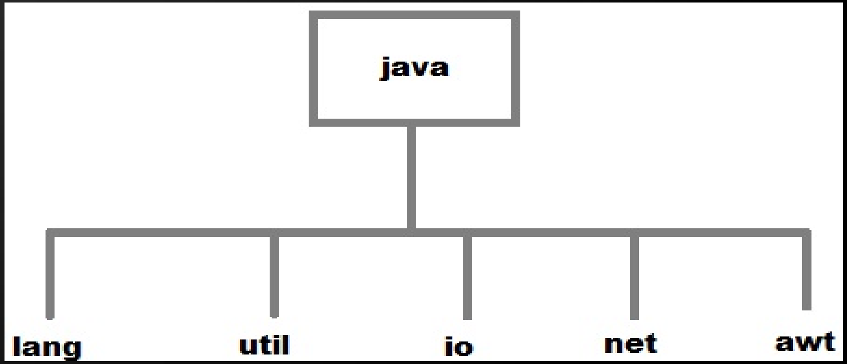
Packages

- A Package can be defined as a grouping of related types (classes, interfaces, enumerations and annotations) providing access protection and name space management
- Creates a new namespace
- Provides access protection
- Categorized into two forms:
  - Built-in Packages such as java, lang, awt, javax, swing, net, io, util, sql, etc.
    - **java.lang** - bundles the fundamental classes
    - **java.io** - classes for input , output functions are bundled in this package
  - User-defined-package:- Java package created by user to categorize classes and interface
- **To compile the Java programs with package statements:**

  ```
  javac -d Destination_folder  file_name.java
  ```

- **To run java package program:**

  ```
  java mypack.Simple
  ```



Packages

- import a package, subpackages will not be imported
- The standard of defining package is domain.company.package e.g. com.xyz.bean or org.sssit.dao
- There can be only one public class in a java source file and it must be saved by the public class name

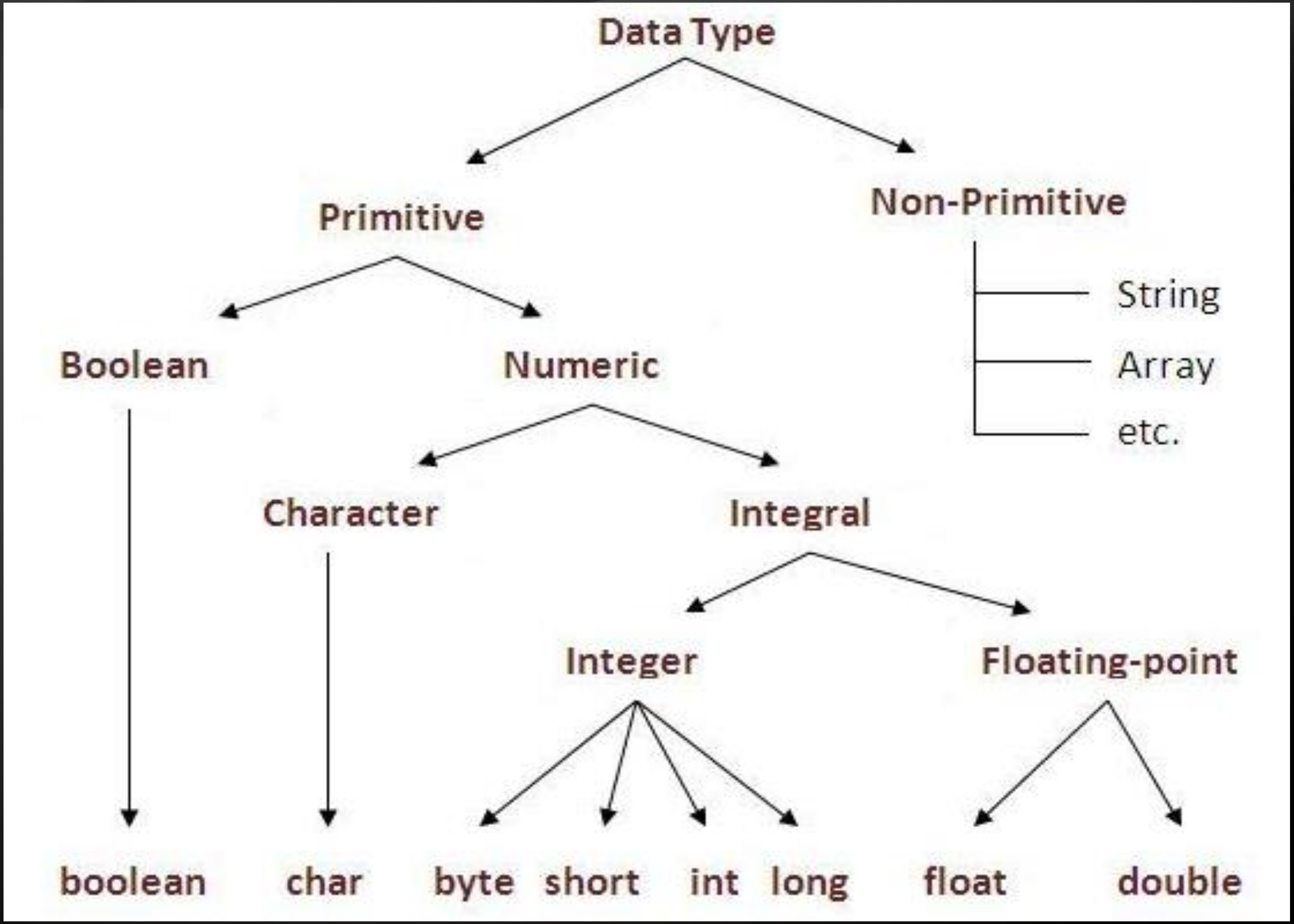
Naming Conventions

Name	Convention
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

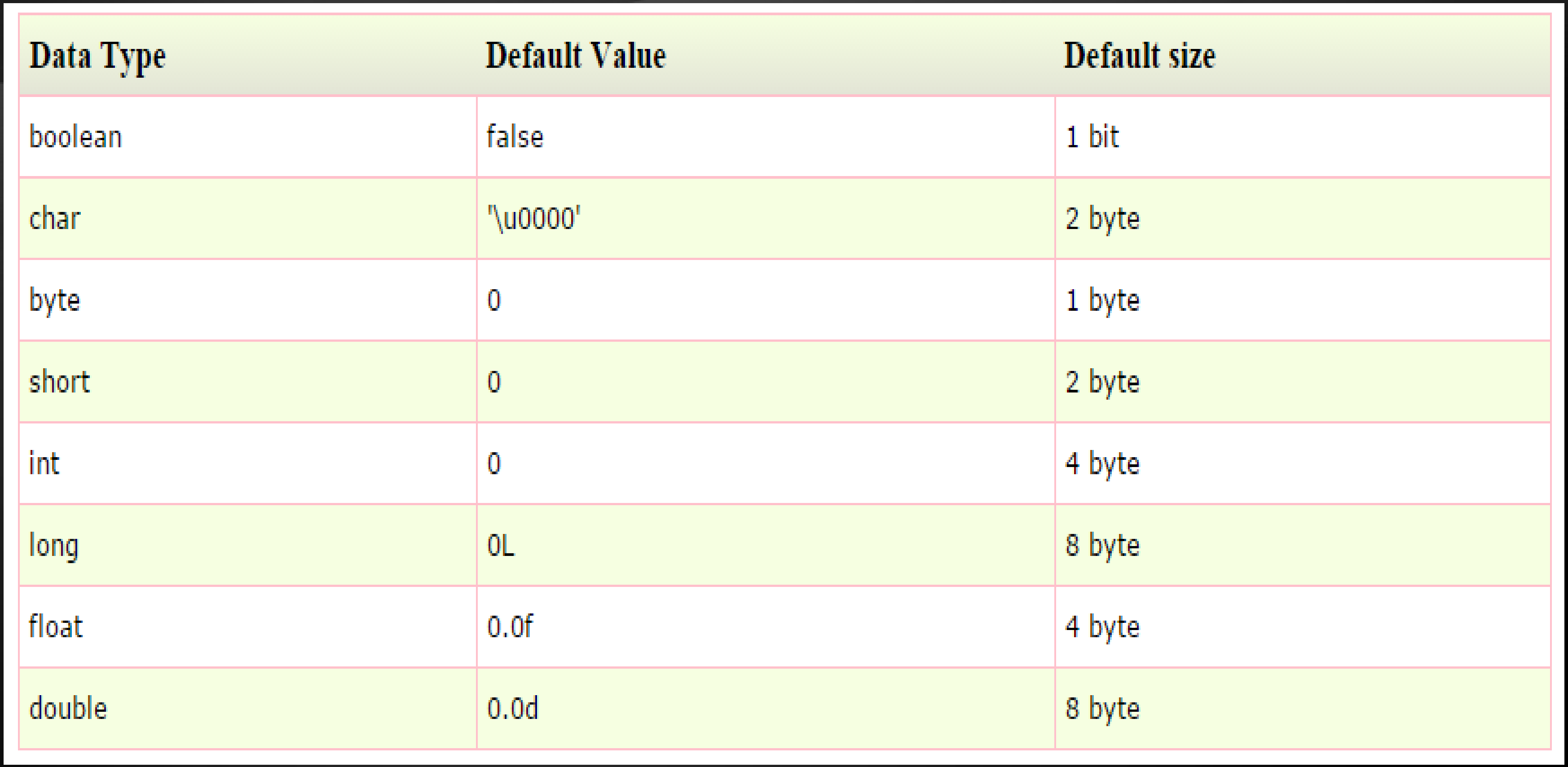
Java Keywords

abstract	assert	boolean	break
byte	case	catch	char
class	const	continue	default
do	double	else	enum
extends	final	finally	float
for	goto	if	implements
import	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	strictfp	super
switch	synchronized	this	throw
throws	transient	try	void
volatile	while		

Data Types in Java



Data Types in Java



Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Operators in Java

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- instanceof Operator - used only for object reference variables
- Misc Operators

- Eg. For instanceof

```
class Vehicle {}

public class Car extends Vehicle {
  public static void main(String args[]){
    Vehicle a = new Car();
    boolean result =  a instanceof Car;
    System.out.println( result );
  }
}
```

Operators in Java

Operators	**Precedence**
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Wrapper Class

- Provides the mechanism *to convert primitive into object and object into primitive*
- All the wrapper classes (Integer, Long, Byte, Double, Float, Short) are subclasses of the abstract class Number
- Since J2SE 5.0, autoboxing and unboxing feature converts primitive into object and object into primitive automatically
- The automatic conversion of primitive into object is known as autoboxing and vice-versa unboxing
- One of the eight classes of java.lang package are known as wrapper class in java

Number Class Methods

Method	Description
compareTo()	Compares this Number object to the argument.
equals()	Determines whether this number object is equal to the argument.
valueOf()	Returns an Integer object holding the value of the specified primitive.
toString()	Returns a String object representing the value of specified int or Integer.
parseInt()	This method is used to get the primitive data type of a certain String.
ceil()	This method is used to get the primitive data type of a certain String.
floor()	Returns the largest integer that is less than or equal to the argument. Returned as a double.
round()	Returns the closest long or int, as indicated by the method's return type, to the argument.
exp()	Returns the base of the natural logarithms, e, to the power of the argument.
log()	Returns the natural logarithm of the argument.
pow()	Returns the value of the first argument raised to the power of the second argument.
sqrt()	Returns the square root of the argument.
sin()	Returns the sine of the specified double value
random()	Returns a random number.

Wrapper Class (Contd...)

Wrapper to Primitive

```
public class WrapperExample1{
public static void main(String args[]){
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally

System.out.println(a+" "+i+" "+j);
}}
```

Primitive to Wrapper

```
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int
int j=a;//unboxing, now compiler will write a.intValue() internally



System.out.println(s1.equals(s3));//true
```

Character Class Methods

isLetter() - Determines whether the specified char value is a letter.

isDigit() - Determines whether the specified char value is a digit.

isWhitespace() Determines whether the specified char value is white space

isUpperCase() - Determines whether the specified char value is uppercase

isLowerCase() - Determines whether the specified char value is lowercase.

toUpperCase() - Returns the uppercase form of the specified char value

toLowerCase() - Returns the lowercase form of the specified char value

toString() - Returns a String object representing the specified character valuethat is, a one-character string

String Class

- The String class is immutable, i.e., once it is created a String object cannot be changed, because java uses the concept of string literal
- To make modifications to Strings of characters, use String Buffer & String Builder Classes

Loop Control

Loops

- While Loop
- Do…While loop
- For loop
- For(declaration : expression) // Enhanced for loop or for each loop

Loop Controls

- Break
- Continue

Decision Making

- If statement
- If…else statement
- Nested if…else statements
- Switch statement
- ? : (Conditional Operator)

Java Array

- It is an object that contains elements of similar data type
- Can store only fixed set of elements
- To make dynamic growth of array at runtime, use Java Collection framework

- dataType[] arr; (or) //declaration
- dataType []arr; (or)
- dataType arr[];

- int a[]=new int[5]; //declaration and instantiation – Allocate space
- **int** a[]={33,3,4,5}; //declaration, instantiation and initialization

- Passing array to method

```
static void min(int arr[]){
………
}
int a[]={33,3,4,5};
min(a);//passing array to method
System.out.println(a.length);
```

Java Array

```
int[][] arr=new int[3][3];
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};      //declaring and initializing 2D array
```

The Arrays Class:

- The java.util.Arrays class contains various static methods for sorting and searching arrays, comparing arrays, and filling array elements. These methods are overloaded for all primitive types.

Exceptions

When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally

Causes:

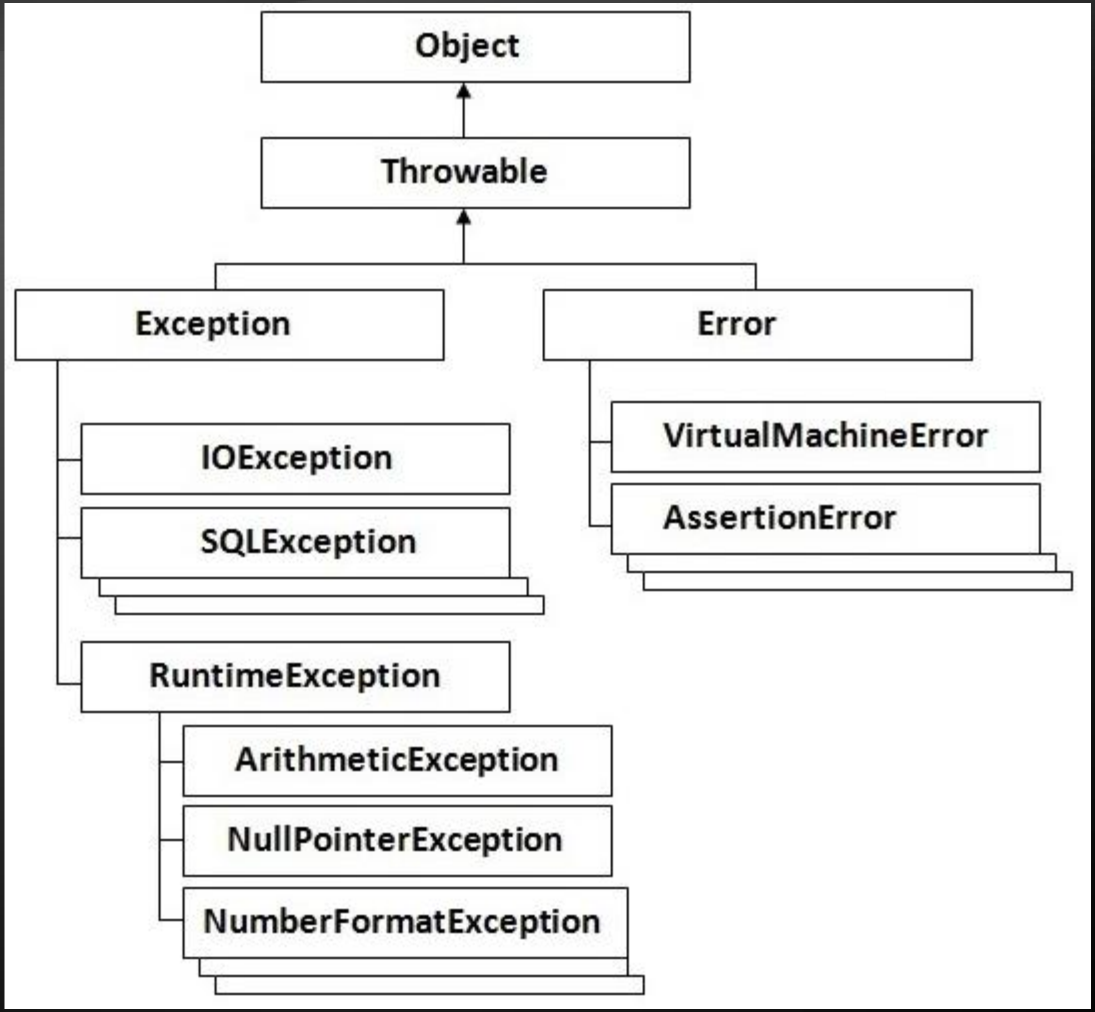
- User error,
- Programmer error
- Physical resources that have failed in some manner
- A file that needs to be opened cannot be found

Categories

- Checked – Occurs at compile time
- Unchecked - Occurs at runtime – E.g.ArrayIndexOutOfBoundsExceptionexception
- Error - These conditions normally happen in case of severe failures, which are not handled by the java programs – E.g Out of Memory

Exceptions

Hierarchy of Java Exception Class



Exceptions

```
import java.io.File;
import java.io.FileReader;

public class FilenotFound_Demo {

   public static void main(String args[]){
      File file=new File("E://file.txt");
      FileReader fr = new FileReader(file);
   }
}
```

```
C:\>javac FilenotFound_Demo.java
FilenotFound_Demo.java:8: error: unreported exception FileNotFoundException; must
be caught or declared to be thrown
      FileReader fr = new FileReader(file);
                      ^
1 error
```

Exceptions

Catching Exceptions

- A method catches an exception using a combination of the try and catch keywords
- A try/catch block is placed around the code that might generate an exception
- Code within a try/catch block is referred to as protected code
- Exception occurred is handled by catch block associated with it

```
try
{
   //Protected code
}catch(ExceptionName e1)
{
   //Catch block
}
```

Exceptions

Multiple Catch Blocks

```
try
{
   //Protected code
}catch(ExceptionType1 e1)
{
   //Catch block
}catch(ExceptionType2 e2)
{
   //Catch block
}catch(ExceptionType3 e3)
{
   //Catch block
}
```

Example:

```
try
{
   file = new FileInputStream(fileName);
   x = (byte) file.read();
}catch(IOException i)
{
   i.printStackTrace();
   return -1;
}catch(FileNotFoundException f) //Not
valid!
{
   f.printStackTrace();
   return -1;
}
```

more than one exceptions can be handled using a single catch block

```
catch (IOException|FileNotFoundException
ex) {
   logger.log(ex);
   throw ex;
```

Exceptions

Throws / throw keywords

- If a method does not handle a checked exception, the method must declare it using the throws keyword. The throws keyword appears at the end of a method's signature
- throws is used to postpone the handling of a checked exception and throw is used to invoke an exception explicitly

```
import java.io.*;
public class className
{
   public void deposit(double amount) throws RemoteException
   {
      // Method implementation
      throw new RemoteException();
   }
   //Remainder of class definition
}
```

Exceptions

Throws / throw keywords

A method can declare that it throws more than one exception, in which case the exceptions are declared in a list separated by commas

```
public void withdraw(double amount) throws RemoteException,
                                   InsufficientFundsException
  {
     // Method implementation
  }
```

The finally block

- Follows a try block or a catch block. A finally block of code always executes, irrespective of occurrence of an Exception

```
try
{
  //Catch block
}catch(ExceptionType3 e3)
{
  //Catch block
}finally
{
  //The finally block always executes.
}
```

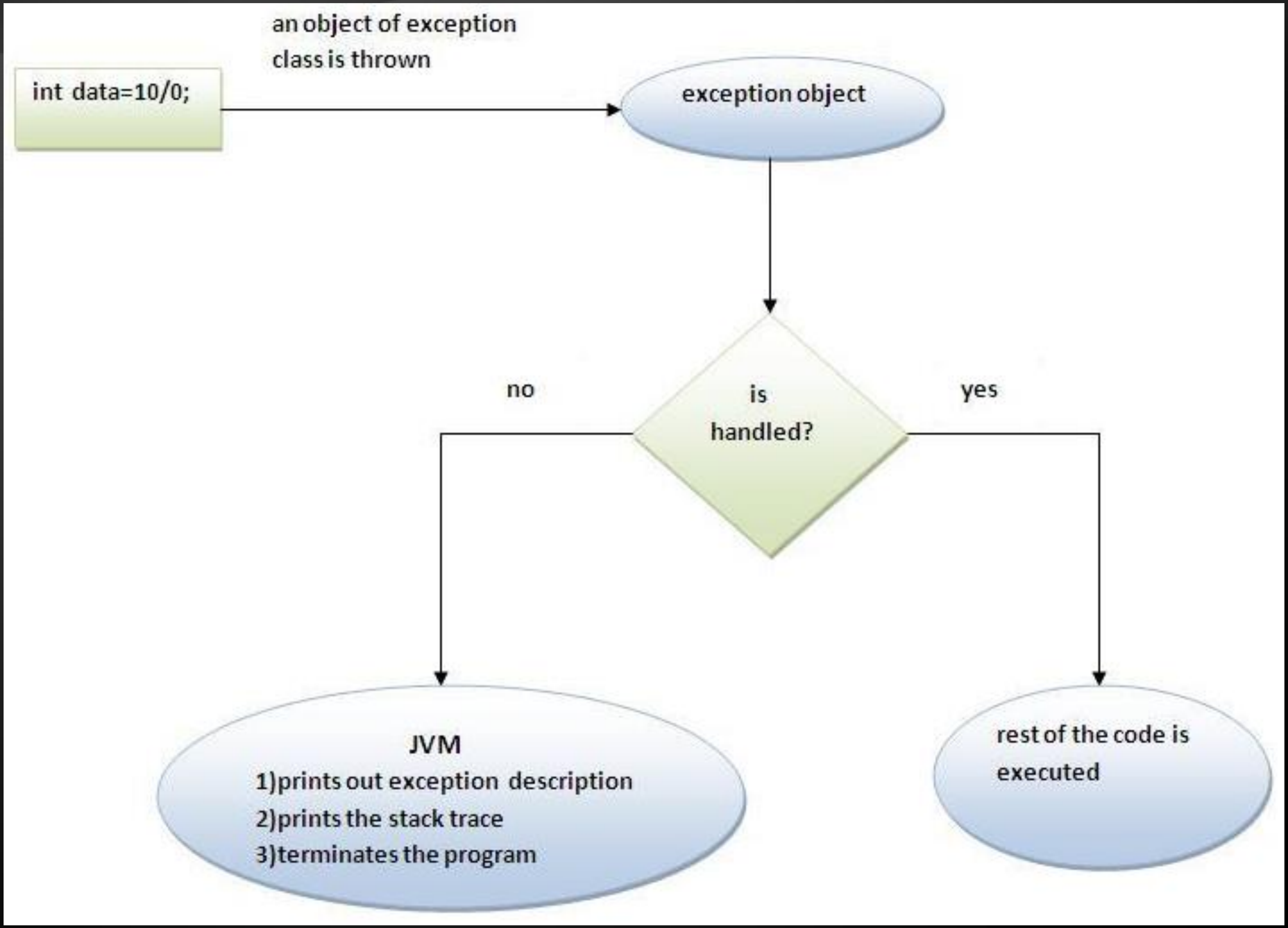
Exceptions

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.

No.	final	finally	finalize
1)	Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be changed.	Finally is used to place important code, it will be executed whether exception is handled or not.	Finalize is used to perform clean up processing just before object is garbage collected.

Exceptions

Internal working of java try-catch block

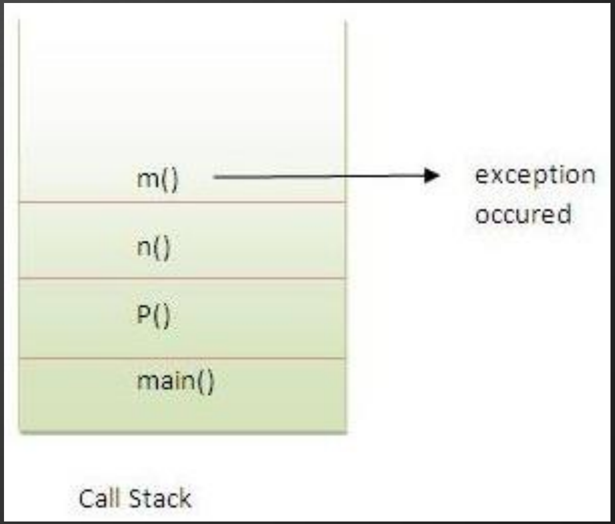

int data=10/0;
an object of exception class is thrown
exception object
is handled?
no
yes
JVM
1)prints out exception description
2)prints the stack trace
3)terminates the program
rest of the code is executed

Custom Exception

```
static void validate(int age)throws InvalidAgeException{
    if(age<18)
     throw new InvalidAgeException("not valid");
    else
     System.out.println("welcome to vote");
  }
```

Exception Propagation

An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method, If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This is called exception propagation.



Exceptions

Keynote:

1. At a time only one Exception is occured and at a time only one catch block is executed
2. All catch blocks must be ordered from most specific to most general i.e. catch for ArithmeticException must come before catch for Exception
3. Finally block in java can be used to put "cleanup" code such as closing a file, closing connection, etc.
4. The finally block will not be executed if program exits(either by calling System.exit() or by causing a fatal error that causes the process to abort)
5. By default Unchecked Exceptions are forwarded in calling chain (propagated), Checked Exceptions are not