gratis exam links:

----------------

http://gratisexam.com/oracle/1z0-808-exam url

https://www.gratisexam.com/oracle/1z0-808-exam

Java Basics

- Define the scope of variables

- Define the structure of a Java class

- Create executable Java applications with a main method; run a Java program from the command line; produce console output

- Import other Java packages to make them accessible in your code

- Compare and contrast the features and components of Java such as: platform independence, object orientation, encapsulation, etc.

Working With Java Data Types

- Declare and initialize variables (including casting of primitive data types)

- Differentiate between object reference variables and primitive variables

- Know how to read or write to object fields

- Explain an Object's Lifecycle (creation, "dereference by reassignment" and garbage collection)

- Develop code that uses wrapper classes such as Boolean, Double, and Integer

Using Operators and Decision Constructs

- Use Java operators; use parentheses to override operator precedence

- Test equality between Strings and other objects using == and equals ()

- Create if and if/else and ternary constructs

- Use a switch statement

Creating and Using Arrays

- Declare, instantiate, initialize and use a one-dimensional array

- Declare, instantiate, initialize and use multi-dimensional arrays

Using Loop Constructs

- Create and use while loops

- Create and use for loops including the enhanced for loop

- Create and use do/while loops

- Compare loop constructs

- Use break and continue

Working with Methods and Encapsulation

- Create methods with arguments and return values; including overloaded methods

- Apply the static keyword to methods and fields

- Create and overload constructors; differentiate between default and user defined constructors

- Apply access modifiers

- Apply encapsulation principles to a class

- Determine the effect upon object references and primitive values when they are passed into methods that change the values

Working with Inheritance

- Describe inheritance and its benefits

- Develop code that makes use of polymorphism; develop code that overrides methods; differentiate between the type of a reference and the type of an object

- Determine when casting is necessary

- Use super and this to access objects and constructors

- Use abstract classes and interfaces

Handling Exceptions

- Differentiate among checked exceptions, unchecked exceptions, and Errors

- Create a try-catch block and determine how exceptions alter normal program flow

- Create and invoke a method that throws an exception

- Recognize common exception classes (such as NullPointerException, ArithmeticException, ArrayIndexOutOfBoundsException, ClassCastException)

Working with Selected classes from the Java API

- Manipulate data using the StringBuilder class and its methods

- Create and manipulate Strings

- Create and manipulate calendar data using classes from java.time.LocalDateTime, java.time.LocalDate, java.time.LocalTime, java.time.format.DateTimeFormatter, java.time.Period

- Declare and use an ArrayList of a given type

- Write a simple Lambda expression that consumes a Lambda Predicate expression

Assume the following:

1. Missing package and import statements:

- If sample code do not include package or import statements, and the question does not explicitly refer to these missing statements, then assume that all sample code is in the same package, or import statements exist to support them.

2. No file or directory path names for classes:

- If a question does not state the file names or directory locations of classes, then assume one of the following, whichever will enable the code to compile and run:

- All classes are in one file

- Each class is contained in a separate file, and all files are in one directory

3. Unintended line breaks:

- Sample code might have unintended line breaks. If you see a line of code that looks like it has wrapped, and this creates a situation where the wrapping is significant (for example, a quoted String literal has wrapped), assume that the wrapping is an extension of the same line, and the line does not contain a hard carriage return that would cause a compilation failure.

4. Code fragments:

- A code fragment is a small section of source code that is presented without its context. Assume that all necessary supporting code exists and that the supporting environment fully supports the correct compilation and execution of the code shown and its omitted environment.

5. Descriptive comments:

- Take descriptive comments, such as "setter and getters go here," at face value. Assume that correct code exists, compiles, and runs successfully to create the described effect.

The Oracle Java SE Programmer I Certification Program certifies candidates on skills and knowledge related to Oracle Java SE products and technologies. The Oracle 1Z0-808 is granted based on a combination of passing exams, training, and performance-based assignments, depending on the level of certification. Java SE 8 Programmer I certification is a real benchmark of experience and expertise that helps you stand out in a crowd among employers. To ensure success, Oracle recommends combining education courses, practice exams, and hands-on experience to prepare for your Oracle Certified Associate Java SE 8 Programmer (OCA) certification exam as questions will test your ability to apply the knowledge you have gained in hands-on practice or professional experience.

============================================================================================================================================================================================

IMP NOTES :

-----------

Identifier :

------------

A name in java program is called an identifier.it may be class Name, method Name, variable name and label name.

- The allowed character for creating an identifer are A-Z,a-z,0-9,\_ and $.

- if we use a ny other characters we will get compile time error.

- Number are not allowed in the beginning of the identifier.

- Java is CaseSensitive langauage.

- There is no length limit for the java identifier but it is not recommended more than 15 length.

- we cant use reserved keywords for identifiers.

- All Predefined class and Interfaces names can be used as an IDENTIFIER.Even though it is legal to use class name and interface name as an identiers but it is not a good programming practice.

Reserved KeyWords(53):

----------------------

In Java certain words are reserved for specific functionality such type of we called reserved words.

Keywords(50) : Used Keywords(48) unused Keywords(2): goto and const

Reserved Literals: true,false and null

ENUM : to define group of named constants.

used Keywords:

datatypes(8) : byte,short,int,long,float,double,char,boolean.

flow control(11): for,while,do while,switch,case,default,break,continue,return,if,else.

modifiers(11) : public,static,final,protected,private,abstract,strictfp,volatile,

transient,native,synchronized.

exception Handling(6) : try,catch,finally,throw,throws,assert.

class related keywords(6) : class,interface,package,import,implements and extends.

object related keywords(4) : new,instanceOf,super and this

NOTE :

-> all reserved words in java are in lowercase.

-> new keywords : enum,assert,strictfp.

-> instanceof,strictfp,const,syncronized,extends,implements,import.

DATA TYPES

-----------

Integral,floating,character and boolean data types we have in java.

->By default every integral literal is int type but we can specify explicitly as long type by suffixing with small "l" (or) capital "L".

->Floating point literal is by default double type but we can specify explicitly as float type by suffixing with f or F.

-> We can specify explicitly floating point literal as double type by suffixing with d or D.

->Every escape character in java acts as a char literal.(\n ,\t,\f,\r,\b,\',\",\\)

-> we can use more than one underscore between digits.

ARRAYS:

--------

->At the time of array creation compulsory we should specify the size otherwise we will get compile time error. int[] a= new int[2];

int[] a1,b1 --> a1-1,b1-1

int[] []a1,b1-->a1-1,b1-1

inr[] a1[],b1;--a1-2,b1-1

int[] a1[],b1[];--invalid declaration.

-> if we want to specify the dimension before the variable that rule is applicable only for 1st variable.second variable onwards we cant apply in the same declaration.

->Every array in java is an object so we can create it by using new operator.

-> For every array type corresponding classes are avaialbel but these classes are a oart of java language but not available to the programmer level.eg [I,[[I,[[D.

->It is legal to have an array with size zero in java.

int[] a = new int[0]; //valid

->If we are taking array size with -ve int value then we will get runtime exception saying NegativeArraySizeException.

-> The allowed data types to specify array size are byte, short, char, int.By mistake if we are using any other type we will get compile time error.

MULTI DIMENSIONAL ARRAY :

--------------------------

in java multi dimensional arrays are implemented as arary of arrays approach but not in the matrix form.main advantage of this approach is improve memory utilization.

-> Array Initialization : when ever we are craeting an array every elements is initialised with default values automatically.

Note : when ever we are printing object reference internally toString() method will be executed which is implemented by default to return the className@hexaDecimalrepresentationOfHashCode.

-> Once we created an array it will be initialised with default values if we are not satisfied with those values we can replace it with our own custiomside values.

-> if we are trying to access array eleents with out of range index we will get runtime exception saying ArrayIndexOutOfBoundException.

Array single line declaration,initialisation in a single line.

int[] a={1,2,3};

->in multi dimension array length variable represents only base size but not total size.

Anonymous Arrays: an array without name is called anonymous array,it is for just instant use ,it will look as follows new int[]{1,2,3,4};

->at the time of array creation we cant specify the size otherwise we will get compile time error.

-> in case of interface type objects as array elements we can provide it implemented class objects.

-> in case of object type arrays as array elements we can provide either declared type objects or its child class objects.

->A char value can be promoted to int type but char array cannot be promoted to int array.

->In the case of object type arrays child type array can be assign to parent type array variable.

eg: String[] a={"a","b"};

Object o=a; //valid

-> when ever we are assigning one array to another array internal elements wont be copied just reference variable will be assigned hence sizes are not important but types must ne matched.

-> when we are assigning one array to another array dimensions must be matched that is in the place of one dimension array we should provide the same type only otherwise we will get compile time error.

Example:

int[][] a=new int[3][];

a[0]=new int[4][5];//C.E:incompatible types(invalid)

a[0]=10;//C.E:incompatible types(invalid)

a[0]=new int[4];//(valid)

Note: Whenever we are performing array assignment the type and diension must be matched but sizes are not important.

Example 1:

int[][] a=new int[3][2];

a[0]=new int[3];

a[1]=new int[4];

a=new int[4][3];

Types Of Variables :

--------------------

Primitive Variables : used to hold primitive values

Reference Variables : used to refer object.

Based on the behaviour and position of declaration all variables are classified into 3 types.

1. Instance Variables :

---------------------------

- If the value of a variable is varied from object to object such type of variables are called instance variables.

- For every object a separate copy of instance variables will be created.

- Instance variables will be created at the time of object creation and destroyed at the time of object destruction hence the scope of instance variables is exactly same as scope of objects.

- Instance variables will be stored on the heap as the part of object.

- Instance variables should be declared with in the class directly but outside of any method or block or constructor.

- Instance variables can be accessed directly from Instance area. But cannot be accessed directly from static area.

- But by using object reference we can access instance variables from static area

Note : For the instance variables it is not required to perform initialization JVM will always provide default values

2. Static variables :

-----------------------

- If the value of a variable is not varied from object to object such type of variables is not recommended to declare as instance variables. We have to declare such

type of variables at class level by using static modifier.

- In the case of instance variables for every object a separate copy will be created but in the case of static variables for entire class only one copy will be created

and shared by every object of that class.

- Static variables will be crated at the time of class loading and destroyed at the time of class unloading hence the scope of the static variable is exactly same as

the scope of the .class file.

- Static variables will be stored in method area. Static variables should be declared with in the class directly but outside of any method or block or constructor

- Static variables can be accessed from both instance and static areas directly.

- We can access static variables either by class name or by object reference but usage of class name is recommended.

- But within the same class it is not required to use class name we can access directly.

For the static variables it is not required to perform initialization explicitly, JVM will always provide default values

3. Local Variables :

- Some times to meet temporary requirements of the programmer we can declare

- variables inside a method or block or constructors such type of variables are called local variables or automatic variables or temporary variables or stack variables.

Local variables will be stored inside stack.

- The local variables will be created as part of the block execution in which it is declared and destroyed once that block execution completes. Hence the scope of the local

variables is exactly same as scope of the block in which we declared

- The local variables will be stored on the stack.

- For the local variables JVM won't provide any default values compulsory we should perform initialization explicitly before using that variable.

- It is never recommended to perform initialization for the local variables inside logical blocks because there is no guarantee of executing that block always at runtime.

- It is highly recommended to perform initialization for the local variables at the time of declaration at least with default values.

Note: The only applicable modifier for local variables is final. If we are using any other modifier we will get compile time error.

->For the instance variables and static variables it is not required to perform initialization JVM will always provide default values

->For the local variables JVM won't provide any default values compulsory we Should perform initialization explicitly before using that variable.

->Instance and static variables can be accessed by multiple Threads simultaneously and hence these are not Thread safe but local variables can be accessed by only one Thread at a time and hence local variables are Thread safe.

-> If we are not declaring any modifier explicitly then it means default modifier but this rule is applicable only for static and instance variables but not local variable.

->The only applicable modifier for local variables is final. If we are using any other modifier we will get compile time error.

VAR-ARGS : where ever we have one dimensional arrays we can replace it with var-args parameter but converse is not possible if we do so we will get compile time error.but it acts as a single dimensional array in method arguments.

-> Once we created an array every element is always initialized with default values irrespective of whether it is static or instance or local array.

-> Every variable in java should be either instance or static or local.

-> Every variable in java should be either primitive or reference

->Every variable in java should be either instance or static or local.

MAIN METHOD :

-------------

- Whether the class contains main() method or not,and whether it is properly declared or not,these checking's are not responsibilities of the compiler, at runtime JVM is responsible for this.

- If JVM unable to find the required main() method then we will get runtime exception

saying NoSuchMethodError: main.

Even though above syntax is very strict but the following changes are acceptable to

main() method.

1. The order of modifiers is not important that is instead of public static we can

take static public.

2. We can declare string[] in any acceptable form

String[] args

String []args

String args[]

3. Instead of args we can use any valid java identifier.

4. We can replace string[] with var-arg parameter.

Example: main(String... args)

5. main() method can be declared with the following modifiers.

final, synchronized, strictfp

Note:

Case 01 : Overloading of the main() method is possible but JVM always calls string[] argument main() method only.

Case 02 : Inheritance concept is applicable for static methods including main() method

hence while executing child class if the child class doesn't contain main() method then

the parent class main() method will be executed.

Command Line Args:

------------------

Within the main() method command line arguments are available in the form of String

hence "+" operator acts as string concatenation but not arithmetic addition.

->Space is the separator between 2 command line arguments and if our command line argument itself contains space then we should enclose with in double quotes.

OPERATOR and ASSIGNMENTS

------------------------

INCREMENT && DECREMENT :

-> In increment and decrement opeations we can apply only for variables but not for constant values.otherwise we will get compile time error. eg: int x=++8; //CE

-> nesting of increment and decrement operation will produce compile time error eg: inty=++(x++);

-> for final variables we cannot perform increment and decrement operation.

-> we can apply increment and decrement for all the primitive data types except boolean.

ARTHIMETIC OPERATIONS:

- if we perform any operation on the operands we will get Max(int,data type a,data type b);

Relational Operators:

------------------------

->We can apply relational operators for every primitive type except boolean.

->We can't apply relational operators for object types.

->Nesting of relational operator is not allowed.

Equality Operators(==,!=)

---------------------------

->We can apply equality operators for every primitive type including boolean type also

->To use the equality operators between object type compulsory these should be some relation between argument types(child to parent , parent to child) ,Otherwise we will get Compiletime error incompatible types

->For any object reference of on r==null is always false , but null==null is always true .

instanceof Operator:

---------------------

->We can use the instanceof operator to check whether the given an object is perticular type or not r instanceof x (here r is an object reference and x is a class or an interface).

->To use instance of operator compulsory there should be some relation between argument types (either child to parent Or parent to child Or same type) Otherwise we will get compile time error saying inconvertible types

->Whenever we are checking the parent object is child type or not by using instanceof operator that we get false.

TypeCasting Operator :

-----------------------

implicit type casting : Higher to lower.// compiler is responsible for conversion at compile time only.

Explicit type casting : higher to lower.// programmer is responsible for conversion.

Assignment Operator :

-----------------------

chained assignment :

int a,b,c,d; a=b=c=d=12;//valid

int a=b=c=d=12;//invalid Compile Time Error.

compound assignment operator : +=,-=,\*=,/=.

Conditional Operator :

----------------------

dataType variable = condition ? true value : false value;

operator precedence :

----------------------

Arthimetic operators : \*,/,%,+,-

Assignment operator : +=,-=,\*,/=,%=.

new vs newInstance;

---------------------

new : if we know the class name at the begining then we can create th eobject by using new

-- for the hard coded class name for the corresponding .class file is not available at the runtime then we will "NoClassDefFoundError" exception we will get.

newInstance() : if we dont know the object at beginning but this will be available at the runtime then we can go by newInstance method.

-- for the dynamically provided class name for the corresponding .class file is not available at the runtime then we will "ClassNotFOundException" exception we will get.

instanceof vs isInstance()

--------------------------

we use instanceof to check whether the given object reference is of particular type or not when we know the object at the begining but if we dont know the object at begining but at the run time then we will use isInstance().

Flow COntrol :

--------------

1. Selection statements : if-else,switch.

2. iterative statements : while(),do-while(),for(),forEach().

3. transfer statements : break,continue,return,try-catch-finally,assert.

if else : it will take boolean as an argument.

------- -bothe else part and curly braces are optional.

-without curly bases we can take only one statement under if but it should not be declarative statement.

Switch:

-------

->Until 1.4 version the allow types for the switch argument are byte, short, char, int but from 1.5 version on wards the corresponding wrapper classes (Byte, Short, Character,

Integer) and "enum" types also allowed.+String also allowed.

\* Curly braces are mandatory.(except switch case in all remaining cases curly braces are optional )

\* Both case and default are optional.

\* Every statement inside switch must be under some case (or) default. Independent statements are not allowed.

\* Every case label should be "compile time constant" otherwise we will get compile time error.

\* But switch argument can be expressions , but case label should be constant expression.

\* Every case label should be within the range of switch argument type.

\* Duplicate case labels are not allowed.

DEfault :

 With in the switch we can take the default only once

 If no other case matched then only default case will be executed

 With in the switch we can take the default any where, but it is convension to take default as last case.

While :

-------

- if we don't know the no of iterations in advance then best loop is while loop

- The argument to the while statement should be Boolean type. If we are using any other type we will get compile time error.

- Curly braces are optional and without curly braces we can take only one statement which should not be declarative statement.

unReachable statement :

-----------------------

 Every final variable will be replaced with the corresponding value by compiler.

 If any operation involves only constants then compiler is responsible to perform that operation.

 If any operation involves at least one variable compiler won't perform that operation. At runtime jvm is responsible to perform that operation.

- Compiler won't check unreachability in the case of if-else it will check only in loops.

DO-while :

----------

- If we want to execute loop body at least once then we should go for do-while

- Curly braces are optional.

Without curly braces we can take only one statement between do and while and it should not be declarative statement.

FOR Loop :

----------

- Curly braces are optional and without curly braces we can take only one statement which should not be declarative statement

- This section will be executed only once.

Here usually we can declare loop variables and we will perform initialization.

We can declare multiple variables but should be of the same type and we can't declare different type of variables.

In initialization section we can take any valid java statement including "s.o.p" also.

Conditional check:

We can take any java expression but should be of the type Boolean.

Conditional expression is optional and if we are not taking any expression compiler will place true.

Increment and decrement section:

Here we can take any java statement including s.o.p also.

All 3 parts of for loop are independent of each other and all optional.

BREAK :

-------

- inside loops,switch case statements and labels we can use break statement.

- These are the only places where we can use break statement. If we are using anywhere else we will get compile time error.

Continue :

----------

- We can use continue statement to skip current iteration and continue for the next iteration.

- We can use continue only inside loops if we are using anywhere else we will get compile time error saying "continue outside of loop".

==========================================================================================================================================================================================

Java Source File Structure:

---------------------------

 A java Program can contain any no. Of classes but at most one class can be

declared as public. "If there is a public class the name of the Program and name

of the public class must be matched otherwise we will get compile time error".

 If there is no public class then any name we gives for java source file.

NOTE :

------

1. If there is no public class then we can use any name for java source file there are no restrictions.

2. If class B declared as public then the name of the Program should be B.java otherwise we will get compile time error saying "class B is public, should be declared in a file named B.java".

3. If both B and C classes are declared as public and name of the file is B.java then we will get compile time error saying "class C is public, should be declared in a file named C.java".

It is highly recommended to take only one class for source file and name of the Program (file) must be same as class name. This approach improves readability

and understandability of the code

COMPILING A JAVA PRAGRAM : javac <program file name>.java

RUNNING A JAVA PROGRAM : java <.dot class file name>

Import Statement :

------------------

- to improve readability of the program from fully qualified names we can go with import statement.

- when ever we are using import statement it is not required to use fully qualified names we can use short names directly.

Types of Import :

------------------

1.Explicit import : import java.util.ArrayList.

2.Implicit import : import java.util.\*;

While resolving class names compiler will always gives the importance in the following

order.

1. Explicit class import

2. Classes present in current working directory.

3. Implicit class import.

-> Whenever we are importing a package all classes and interfaces present in that package are by default available but not sub package classes.

-> In any java Program the following 2 packages are not require to import because these are available by default to every java Program.

1. java.lang package

2. default package(current working directory)

-> "Import statement is totally compile time concept" if more no of imports are there then more will be the compile time but there is "no change in execution time".

Static Import :

----------------

-> Usually we can access static members by using class name but whenever we are using static import it is not require to use class name we can access directly.

But 2 classes or interfaces can contain a method or variable with the same name is very

common hence ambiguity problem is also very common in static import.

While resolving static members compiler will give the precedence in the following order.

1. Current class static members

2. Explicit static import

3. implict static import

import static java.lang.Math.sqrt;

import static java.lang.Math.\*;

What is the difference between general import and static import ?

 We can use normal imports to import classes and interfaces of a package.

whenever we are using normal import we can access class and interfaces directly

by their short name it is not require to use fully qualified names.

 We can use static import to import static members of a particular class.

whenever we are using static import it is not require to use class name we can

access static members directly.

Package Statement :

-------------------

it is an encapsulation mechanism to group related classes and interfaces into a single module.

How to compile package program ? // javac -d . <program>.java

how to run java program ? //java <package path >. java program .

-> In any java Program the 1st non comment statement should be package statement [if it is available] otherwise we will get compile time error.

-> An empty source file is a valid java Program.

Java Source File Structure :

----------------------------

Package-- one for every class.

import -- any number in a file

class/interface/enum --anu number.

Class Modifiers :

-----------------

The only applicable modifiers for Top Level classes are:

1. Public

2. Default

3. Final

4. Abstract

5. Strictfp

If we are using any other modifier we will get compile time error.

But For the inner classes the following modifiers are allowed.

1. Public

2. Default

3. Final

4. Abstract

5. Strictfp

+

private

protected

static

PUBLIC : If a class declared as public then we can access that class from anywhere. With in the package or outside the package.

DEFAULT : If a class declared as the default then we can access that class only within the current package hence default access is also known as "package level access".

FINAL : Final is the modifier applicable for classes, methods and variables.

 Whatever the methods parent has by default available to the child.

 If the child is not allowed to override any method, that method we have to declare with final in parent class. That is final methods cannot overridden.

 If a class declared as the final then we cann't creates the child class that is inheritance concept is not applicable for final classes.

Note: Every method present inside a final class is always final by default whether we are declaring or not. But every variable present inside a final class need not be final.

The main advantage of final keyword is we can achieve security.

Whereas the main disadvantage is we are missing the key benefits of oops:

polymorsim (because of final methods), inheritance (because of final classes) hence if there is no specific requirement never recommended to use final keyboard

ABSTRACT : Abstract is the modifier applicable only for methods and classes but not for variables.

Even though we don't have implementation still we can declare a method with abstract modifier.

That is abstract methods have only declaration but not implementation.

Hence abstract method declaration should compulsory ends with semicolon.

below are the illgal combination by the abstract

abstract -- static,private,syncronised,strictfp,final,native.

For any java class if we are not allow to create an object such type of class we have to declare with abstract modifier that is for abstract class instantiation is not possible.

What is the difference between abstract class and abstract method ?

 If a class contain at least on abstract method then compulsory the corresponding class should be declare with abstract modifier. Because implementation is not complete and hence we can't create object of that class.

 Even though class doesn't contain any abstract methods still we can declare the class as abstract that is an abstract class can contain zero no of abstract methods also.

 If a class extends any abstract class then compulsory we should provide implementation for every abstract method of the parent class otherwise we have to declare child class as abstract.

->Final class cannot contain abstract methods whereas abstract class can contain final method.

STRICTFP :

 strictfp is the modifier applicable for methods and classes but not for variables.

 Strictfp modifier introduced in 1.2 versions.

 Usually the result of floating point of arithmetic is varing from platform to platform , to overcome this problem we should use strictfp modifier.

 If a method declare as the Strictfp then all the floating point calculations in that method has to follow IEEE754 standard, So that we will get platform independent results.

-> But we can declare a class with abstract and strictfp modifier simultaneously.That is abstract strictfp combination is legal for classes but illegal for methods.

PRIVATE :

If a member declared as the private then we can access that member only with in the current class.

 Private methods are not visible in child classes where as abstract methods should be visible in child classes to provide implementation hence private, abstract combination is illegal for methods.

PROTECTED :

 If a member declared as the protected then we can access that member within the current package anywhere but outside package only in child classes.Protected=default+kids.

 We can access protected members within the current package anywhere either by child reference or by parent reference

 But from outside package we can access protected members only in child classes and should be by child reference only that is we can't use parent reference to call protected members from outside package.

Native :

--------

Native is a modifier applicable for methods but not for classes and variables.

the methods which are implemented in non java are called native or foreign methods.

FINAL INSTANCE VARIABLE :

-> If the instance variable declared as the final compulsory we should perform initialization explicitly and JVM won't provide any default values.whether we are using or not otherwise we will get compile time error.

->For the final instance variables we should perform initialization before constructor completion. That is the following are various possible places for this.

we have to initialise them at

1.at the time of declaration.

2.inside instance block.

3.inside constructor if we do anywhere apart from these places we will get compile time error.

STATIC FINAL VARIABLE:

->If the static variable declare as final then compulsory we should perform initialization explicitly whether we are using or not otherwise we will get compile time error.(The JVM won't provide any default values)

->For the final static variables we should perform initialization before class loading completion otherwise we will get compile time error. That is the following are possible places.

we have to initilise them at declaration time and at static block.

Final local variables:

 To meet temporary requirement of the Programmer sometime we can declare the variable inside a method or block or constructor such type of variables are called local variables.

 For the local variables jvm won't provide any default value compulsory we should perform initialization explicitly before using that variable.

Note: The only applicable modifier for local variables is final if we are using any other modifier we will get compile time error.

 For instance and static variables JVM will provide default values but if instance

and static declared as final JVM won't provide default value compulsory we

should perform initialization whether we are using or not .

 For the local variables JVM won't provide any default values we have to

perform explicitly before using that variables , this rule is same whether local

variable final or not

STATIC MODIFIER :

-----------------

Static is the modifier applicable for methods, variables and blocks.

 We can't declare a class with static but inner classes can be declaring as the static.

 In the case of instance variables for every object a separate copy will be created but in the case of static variables a single copy will be created at class level and shared by all objects of that class.

INTERFACE:

----------

-> Any service requirement specification we can called as INTERFCE.

->Whenever we are implementing an interface compulsory for every method of that interface we should provide implementation otherwise we have to declare class as abstract in that case child class is responsible to provide implementation for remaining methods.

->Whenever we are implementing an interface method compulsory it should be declared as public otherwise we will get compile time error.

->A class can extends only one class at a time.

->A class can implements any no. Of interfaces at a time.

->A class can extend a class and can implement any no. Of interfaces simultaneously.

->An interface can extend any no. Of interfaces at a time.

-> extends followed by implements

-> Every method present inside the interface are always public and abstract by default whether we declared or not

-> As every interface method are always public and abstract so we cannot use the following modifiers for interface methods. // private,static,final,native,strictfp,synchronized,protected

Interface Variables:

--------------------

-> An interface can contains variable.

-> the main purpose of the interface variables is to define requirement level constant.

-> every interface variables is always public static and final whether we declared or not.

public: To make it available for every implementation class.

static: Without existing object also we have to access this variable.

final: Implementation class can access this value but cannot modify.

 As every interface variable by default public static final we can't declare with the

following modifiers.

o Private

o Protected

o Transient

o Volatile

 For the interface variables compulsory we should perform initialization at the

time of declaration only otherwise we will get compile time error.

->Interface variables can be access from implementation class but cannot be modified.

->->If two interfaces contain a method with same signature return types in the

implementation class only one method implementation is enough.

-> if two interfaces contain a method with same name but different arguments in the

implementation class we have to provide implementation for both methods and these

methods acts as a overloaded methods

->If two interfaces contain a method with same signature but different return types then it is not possible to implement both interfaces simultaneously.

->Is a java class can implement any no. Of interfaces simultaneously ?

Yes, except if two interfaces contains a method with same signature but different return types.

Variable Naming Conflicts:

--------------------------

Two interfaces can contain a variable with the same name and there may be a chance

variable naming conflicts but we can resolve variable naming conflicts by using

interface names

Marker Interface :

------------------

If an interface doesn't contain any methods and by implementing that interface if our

objects will get some ability such type of interfaces are called Marker interface (or) Tag

interface (or) Ability interface.

eg: serializable,clonable,RandomAccess.

Without having any methods in marker interface how objects will get ability ?

Internally JVM is responsible to provide required ability.

Why JVM is providing the required ability in marker interfaces ?

To reduce complexity of the programming

Adapter class:

 Adapter class is a simple java class that implements an interface only with empty

implementation for every method.

 If we implement an interface directly for each and every method compulsory we

should provide implementation whether it is required or not. This approach

increases length of the code and reduces readability

We can resolve this problem by using adapter class.

 Instead of implementing an interface if we can extend adapter class we have to

provide implementation only for required methods but not for all methods of

that interface.

 This approach decreases length of the code and improves readability.

public class Test extend AdapterX{{

public void m3(){

}}

 If we don't know anything about implementation just we have requirement specification then we should go for interface.

 If we are talking about implementation but not completely (partial implementation) then we should go for abstract class.

 If we are talking about implementation completely and ready to provide service then we should go for concrete class.

abstract class constructor will be executed when ever we are creating child class object to perform initialization of child object.

CONSTRUCTOR:

------------

The Purpose of Constructor is

 To Initialize an Object but Not to Create an Object.

 Whenever we are creating an Object after Object Creation automatically

Constructor will be executed to Initialize that Object.

 Object Creation by New Operator and then Initialization by Constructor

-> Before Constructor Only Object is Ready and Hence within the Constructor we can

Access Object Properties Like Hash Code.

-> Whenever we are creating Child Class Object automatically Parent Constructor will

be executed but Parent Object won’t be created

===========================================================================================

OOPS:

------

DATA HIDING :

-------------

Our internal data should not go out directly that is outside person can't access our internal data directly.

 By using private modifier we can implement data hiding.

 The main advantage of data hiding is security.

ABSTRACTION :

-------------

Hide internal implementation and just highlight the set of services, is called abstraction.

 By using abstract classes and interfaces we can implement abstraction

ENCAPSULATION:

---------------

Binding of data and corresponding methods into a single unit is called

Encapsulation .

 If any java class follows data hiding and abstraction such type of class is said to be encapsulated class.

Encapsulation=Datahiding+Abstraction

 Every data member should be declared as private and for every member we have to maintain getter & Setter methods.

Tightly encapsulated class :

-----------------------------

A class is said to be tightly encapsulated if and only if every variable of that class

declared as private whether the variable has getter and setter methods are not , and

whether these methods declared as public or not, these checkings are not required to

perform.

IS- A Relationship (Inheritance):

---------------------------------

1. Also known as inheritance.

2. By using "extends" keywords we can implement IS-A relationship.

3. The main advantage of IS-A relationship is reusability.

\*\*1. Whatever the parent has by default available to the child but whatever the child has by default not available to the parent. Hence on the child reference we can call both parent and child class methods. But on the parent reference we can call only methods available in the parent class and we can't call child specific methods.

\*\*2. Parent class reference can be used to hold child class object but by using that reference we can call only methods available in parent class and child specific methods we can't call.

\*\*3. Child class reference cannot be used to hold parent class object.

MIULTIPLE INHERITANCE :

-----------------------

-> Having more than one Parent class at the same level is called multiple inheritance.

-> Any class can extends only one class at a time and can't extends more than one class simultaneously hence java won't provide support for multiple inheritance.

-> But an interface can extends any no. Of interfaces at a time hence java provides support for multiple inheritance through interfaces.

-> Why java won't provide support for multiple inheritance?

There may be a chance of raising ambiguity problems.

-> Why ambiguity problem won't be there in interfaces?

Interfaces having dummy declarations and they won't have implementations hence no

ambiguity problem.

HAS-A RELATIONSHIP:

-------------------

1. HAS-A relationship is also known as composition (or) aggregation.

2. There is no specific keyword to implement HAS-A relationship but mostly we can use new operator.

3. The main advantage of HAS-A relationship is reusability.

4. The main dis-advantage of HAS-A relationship increases dependency between the components and creates maintains problems.

Composition:

------------

Without existing container object if there is no chance of existing contained objects then

the relationship between container object and contained object is called composition

which is a strong association eg: University and departments.

Aggregation :

-------------

Without existing container object if there is a chance of existing contained objects such

type of relationship is called aggregation. In aggregation objects have weak association

eg: University and professors.

Note :

In composition container , contained objects are strongly associated, and but container

object holds contained objects directly

But in Aggregation container and contained objects are weakly associated and

container object just now holds the reference of contained objects.

Method signature:

-----------------

 In java, method signature consists of name of the method followed by argument types.

 In java return type is not part of the method signature.

 Compiler will use method signature while resolving method calls.

 Within the same class we can't take 2 methods with the same signature otherwise we will get compile time error.

POLYMORPHISM:

-------------

- Same name with different forms is the concept of polymorphism.

- We can use the parent reference to hold any child objects.

- We can use the same List reference to hold ArrayList object, LinkedList object, Vector object, or Stack object.

Polymorphism is called into two types they are

1.compiletime/earlyBinding/static: overloading and method hiding.

2.Runtime/late/dynamic : overriding.

1) Inheritance talks about reusability.

2) Polymorphism talks about flexibility.

3) Encapsulation talks about security.

Beautiful definition of polymorphism:

A boy starts love with the word friendship, but girl ends love with the same word

friendship, word is the same but with different attitudes. This concept is nothing but

polymorphism.

Overloading

OVERLOADING :

---------------

- Two methods are said to be overload if and only if both having the same name but different argument types.

- In overloading compiler is responsible to perform method resolution(decision) based on the reference type(but not based on run time object). Hence overloading is also considered as compile time polymorphism(or) static polymorphism (or)early biding.

AUTOMATIC PROMOTION IN OVERLOADING :

------------------------------------

overloading if compiler is unable to find the method with exact match we won't get any compile time error immediately.

 1st compiler promotes the argument to the next level and checks whether the matched method is available or not if it is available then that method will be considered if it is not available then compiler promotes the argument once again to the next level. This process will be continued until all possible promotions still if the matched method is not available then we will get compile time error. This process is called automatic promotion in overloading.

\*\* While resolving overloaded methods exact match will always get high priority,While resolving overloaded methods child class will get the more priority than parent class

 In general var-arg method will get less priority that is if no other method matched then only var-arg method will get chance for execution it is almost same as default case inside switch.

Note : In overloading method resolution is always based on reference type and runtime object won't play any role in overloading.

OVERRIDING :

------------

1. Whatever the Parent has by default available to the Child through inheritance, if the Child is not satisfied with Parent class method implementation then Child is allow to redefine that Parent class method in Child class in its own way this process is called overriding.

2. The Parent class method which is overridden is called overridden method.

3. The Child class method which is overriding is called overriding method.

In overriding method resolution is always takes care by JVM based on runtime object hence overriding is also considered as runtime polymorphism or dynamic polymorphism or late binding.

The process of overriding method resolution is also known as dynamic method dispatch.

Note: In overriding runtime object will play the role and reference type is dummy.

RULES :

-------

1. In overriding method names and arguments must be same.That is method signature must be same.

2. Until 1.4 version the return types must be same but from 1.5 version onwards covariant return types are allowed.

3. According to this Child class method return type need not be same as Parent class method return type its Child types also allowed.

-> Co-variant return type concept is applicable only for object types but not for primitives.

-> Private methods are not visible in the Child classes hence overriding concept is not applicable for private methods. Based on own requirement we can declare the same Parent class private method in child class also. It is valid but not overriding.

eg: object --> String , Number ---> integer //valid

String ---> Object, double---> int //invalid.

-> Parent class final methods we can't override in the Child class.

-> Parent class non final methods we can override as final in child class. We can override native methods in the child classes.

-> We should override Parent class abstract methods in Child classes to provide implementation.

-> We can override a non-abstract method as abstract this approach is helpful to stop availability of Parent method implementation to the next level child classes.

-> Synchronized, strictfp, modifiers won't keep any restrictions on overriding.

final--> no final //invalid

non final--> final //valid

abstract --> non abstarct //valid

nonabstract --> abstract //valid

strictfp-----> nonstrictfp

<-----

synchronized ----> non synchronized

<----

native ---> non native

<--

-> While overriding we can't reduce the scope of access modifier in the child class otherwise we will get compile time errors.

public--> public

protected ---> protected/public

default ---> default/protected/public

private --> overriding not possible.

Checked and Unchecked Exceptions in overriding:

-----------------------------------------------

While overriding if the child class method throws any checked exception compulsory the parent class method should throw the same checked exception or its parent otherwise we will get compile time error.

But there are no restrictions for un-checked exceptions we dont need to handle in the parent method it means.

lly if parent class method is throwing some exception but it is should /shouldnot be handled in the child class method.no issues.

OVERRIDING STATIC METHODS:

--------------------------

-> We can't override a static method as non static.

-> Similarly we can't override a non static method as static.

->It is valid. It seems to be overriding concept is applicable for static methods but it is not overriding it is method hiding.

-> All rules of method hiding are exactly same as overriding except the following differences.

-> A var-arg method should be overridden with var-arg method only. If we are trying to override with normal method then it will become overloading but not overriding.

Overriding with respect to variables:

-------------------------------------

 Overriding concept is not applicable for variables.

 Variable resolution is always takes care by compiler based on reference type.

Note: In the above program Parent and Child class variables, whether both are static or non static whether one is static and the other one is non static there is no change in the answer.

Note:

1. In overloading we have to check only method names (must be same) and arguments (must be different) the remaining things like return type extra not required to check.

2. But In overriding we should compulsory check everything like method names,arguments, return types, throws keyword, modifiers etc.

IIQ : In how many ways we can create an object ? (or) In

how many ways get an object in java ?

1. By using new Operator :

2. Test t = new Test();

3. By using newInstance() :(Reflection Mechanism)

4. Test t=(Test)Class.forName("Test").newInstance();

5. By using Clone() :

6. Test t1 = new Test();

7. Test t2 = (Test)t1.Clone();

8. By using Factory methods :

9. Runtime r = Runtime.getRuntime();

10. DateFormat df = DateFormat.getInstance();

11. By using Deserialization :

12. FileInputStream fis = new FileInputStream("abc.ser");

13. ObjectInputStream ois = new ObjectInputStream(fis);

14. Test t = (Test)ois.readObject();

Constructor :

------------

1. Object creation is not enough compulsory we should perform initialization then

only the object is in a position to provide the response properly.

2. Whenever we are creating an object some piece of the code will be executed

automatically to perform initialization of an object this piece of the code is

nothing but constructor.

3. Hence the main objective of constructor is to perform initialization of an object.

Constructor Vs instance block:

------------------------------

1. Both instance block and constructor will be executed automatically for every object creation but instance block 1st followed by constructor.

2. The main objective of constructor is to perform initialization of an object.

3. Other than initialization if we want to perform any activity for every object creation we have to define that activity inside instance block.

4. Both concepts having different purposes hence replacing one concept with another concept is not possible.

5. Constructor can take arguments but instance block can't take any arguments hence we can't replace constructor concept with instance block.

6. Similarly we can't replace instance block purpose with constructor.

Rules to write constructors:

----------------------------

1. Name of the constructor and name of the class must be same.

2. Return type concept is not applicable for constructor even void also by mistake if

we are declaring the return type for the constructor we won't get any compile time error and runtime error compiler simply treats it as a method.

3.It is legal (but stupid) to have a method whose name is exactly same as class name.

4.The only applicable modifiers for the constructors are public, default, private,protected.If we are using any other modifier we will get compile time error.

Default constructor:

--------------------

1. For every class in java including abstract classes also constructor concept is

applicable.

2. If we are not writing at least one constructor then compiler will generate default

constructor.

3. If we are writing at least one constructor then compiler won't generate any

default constructor. Hence every class contains either compiler generated

constructor (or) programmer written constructor but not both simultaneously.

Prototype of default constructor:

--------------------------------

1. It is always no argument constructor.

2. The access modifier of the default constructor is same as class modifier. (This

rule is applicable only for public and default).

3. Default constructor contains only one line. super(); it is a no argument call to

super class constructor.

SUPER() VS THIS():

------------------

-> The 1st line inside every constructor should be either super() or this() if we are not writing anything compiler will always generate super().

-> We have to take super() (or) this() only in the 1st line of constructor. If we are taking anywhere else we will get compile time error

-> We can use either super() (or) this() but not both simultaneously

-> We can use super() (or) this() only inside constructor. If we are using anywhere else we will get compile time error.

-> THIS() -- That is we can call a constructor directly from another constructor only.

-> We can use these to invoke super class & current constructors directly

SUPER AND THIS : We can use anywhere (i.e., instance area) except static area , other wise we will get compile time error .We can use refers parent class and current

class instance members.

Overloaded constructors :

-------------------------

-> A class can contain more than one constructor and all these constructors having the same name but different arguments and hence these constructors are considered as overloaded constructors.

 Parent class constructor by default won't available to the Child. Hence Inheritance concept is not applicable for constructors and hence overriding concept also not applicable to the constructors. But constructors can be overloaded.

 We can take constructor in any java class including abstract class also but we can't take constructor inside interface.

\*\* Recursive method call is always runtime exception where as recursive constructor invocation is a compile time error.

Note: Compiler is responsible for the following checkings.

----------------------------------------------------------

1. Compiler will check whether the programmer wrote any constructor or not. If

he didn't write at least one constructor then compiler will generate default

constructor.

2. If the programmer wrote any constructor then compiler will check whether he

wrote super() or this() in the 1st line or not. If his not writing any of these

compiler will always write (generate) super().

3. Compiler will check is there any chance of recursive constructor invocation. If

there is a possibility then compiler will raise compile time error.

 If the Parent class contains any argument constructors while writing Child classes we should takes special care with respect to constructors.

 Whenever we are writing any argument constructor it is highly recommended to write no argument constructor also.

\*\* If Parent class constructor throws some checked exception compulsory Child class constructor should throw the same checked exception (or) its Parent. (TOP-> BOTTOM I:E FROM PARENT CONSTRUCTOR TO CHILD CONSTRUCTOR).

\*\* But in case of overriding if child implemented method throws some method compulsory place class method has to throw same or its parent exception..

Singleton Class:

-----------------

For any java class if we allow to create only one ibject such type of classes is said to be singleton classes.

for this we need private constructor,static variable and factory method.

Factory Method:

----------------

By using class name if we are calling a method and that method is returning the class object such type of method we called it as Factory methods.

we are not allowed to create child class but class is not final , How it is Possible ?

By declaring every constructor has private.

class Parent {

private Parent() {

}

We can't create child class for this class

Static COntrol FLow :

----------------------

1.identification of static members from top to bottom in a program.

2.execution static variable assignment and static block from top to bottom.

3. execution of main method.

Instance Control Flow :

-----------------------

Whenever we are executing a java class static control flow will be executed. In the Static

control flow Whenever we are creating an object the following sequence of events will

be performed automatically.

1. Identification of instance members from top to bottom(3 to 8).

2. Execution of instance variable assignments and instance blocks from top to

bottom(9 to 14).

3. Execution of constructor.

Note: static control flow is one time activity and it will be executed at the time of class

loading.

But instance control flow is not one time activity for every object creation it will be

executed.

Whenever we are creating child class object the following sequence of events will be

executed automatically.

1. Identification of instance members from Parent to Child.

2. Execution of instance variable assignments and instance block only in Parent

class.

3. Execution of Parent class constructor.

4. Execution of instance variable assignments and instance blocks in Child class.

5. Execution of Child class constructor.

Note: Object creation is the most costly operation in java and hence if there is no

specific requirement never recommended to crate objects

TYPE CASTING :

--------------

Parent class reference can be used to hold Child class object but by using that reference we can't call Child specific methods

Similarly we can use interface reference to hold implemented class object.

A b= (C) d;

A -- class/Interface

b -- refeerenc variable

C -- class / interface

d-- name of the reference variable or object reference.

Compile time checking :

Rule 1: The type of "d" and "c" must have some relationship [either Child to Parent

(or) Parent to Child (or) same type] otherwise we will get compile time error saying

inconvertible types.

Rule 2: "C" must be either same (or) derived type of "A" otherwise we will get compile

time error saying incompatible types.

Rule 3 : The underlying object type of "d" must be either same (or) derived type of "C"

otherwise we will get runtime exception saying ClassCastException.

Through Type Casting just we are converting the type of object but not object itself that is we are performing type casting but not object casting

Through Type Casting we are not create any new objects for the existing objects we are providing another type of reference variable(mostly Parent type).

eg: String s = new String("Mahendra");

Object o= (Object)o;

the above code is equalent to Object 0=new String("Mahendra");

Coupling:

Cohesion:

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

============================================================================================================================================================================================

EXCEPTION HANDLING

---------------------

Exception : an unwanted and unexpected event that disturb the normal flow of execution is called exception.

The main objective of exception handling is graceful (normal) termination of the program.

Exception handlind doesnt mean that repairing an exception.we have to define an alternative way to continue rest of the program normally.the way of defining alternative is nothing but exception handling.

For every thread JVM will create a separate stack at the time of Thread creation. All Method calls performed by that thread will be stored in that stack. Each entry in the stack is called "Activation record" (or) "stack frame".

After completing every method call JVM removes the corresponding entry from the stack. After completing all method calls JVM destroys the empty stack and terminates the program normally.

Default Exception Handling in Java:

-------------------------------------

1. If an exception raised inside any method then that method is responsible to create Exception object with the following information.

1. Name of the exception.

2. Description of the exception.

3. Location of the exception.(StackTrace)

2. After creating that Exception object, the method handovers that object to the JVM.

3. JVM checks whether the method contains any exception handling code or not. If method won't contain any handling code then JVM terminates that method abnormally and removes corresponding entry form the stack.

4. JVM identifies the caller method and checks whether the caller method contain any handling code or not. If the caller method also does not contain handling code then JVM terminates that caller method also abnormally and removes corresponding entry from the stack.

5. This process will be continued until main() method and if the main() method also doesn't contain any exception handling code then JVM terminates main() method also and removes corresponding entry from the stack.

6. Then JVM handovers the responsibility of exception handling to the default exception handler.

7. Default exception handler just print exception information to the console in the following format and terminates the program abnormally.

Exception hierarchy :

---------------------

Throwable acts as a root for exception hierarchy.

Throwable class contains the following two child classes.

1. Exception: most of the exceptions are caused by our program and these are recoverable.

2. Error : most of the cases errors are not caused by our program these are due to system lack of resource and these are not non recoverable.

Checked Exception : the exceptions which are checked by the compiler whether the programmer handling the exceptions or not , for smooth execution of the program at the run time such type of exceptions are called checked exceptions eg: filenotfound exception.

unchecked exception : the exceptions which are not checked by the compiler at compile time such type of exceptions are called unchecked exceptions eg: runtime exceptions nd errors comes under unchecked exception.

Note : whether the exception is checked or unchecked the exception should occur at runtime only it will not occur at compile time.

Customised Exception Handling by using try - catch :

----------------------------------------------------

Note : 1. Within the try block if anywhere an exception raised then rest of the try block won't be executed even though we handled that exception. Hence we have to place/take only risk code inside try block and length of the try block should be as less as possible.

2.If any statement which raises an exception and it is not part of any try block then it is always abnormal termination of the program.

3. There may be a chance of raising an exception inside catch and finally blocks also in addition to try block.

Various Methods to print exception information :

------------------------------------------------

Throwable class defines the following methods to print exception information to the console.

1. printStackTrace() : - it print name,description and location of the exception.

2. toSTring() : - it prints name and description of the exception.

3. getMessage() : - it prints only description of the exception.

Default exception handler uses printStackTrace to print exception handling info to the console.

Try with Multiple catch blocks:

--------------------------------

The way of handling an exception is varied from exception to exception. Hence for every exception type it is recommended to take a separate catch block. That is try with multiple catch blocks is possible and recommended to use.

Note : If try with multiple catch blocks present then order of catch blocks is very important. It should be from child to parent by mistake if we are taking from parent to child then we will get Compile time error saying "Exception xxx has already been caught."

Finally Block :

---------------

 It is not recommended to take clean up code inside try block because there is no guarantee for the execution of every statement inside a try.

 It is not recommended to place clean up code inside catch block because if there is no exception then catch block won't be executed.

 We require some place to maintain clean up code which should be executed always irrespective of whether exception raised or not raised and whether handled or not handled. Such type of best place is nothing but finally block.

 Hence the main objective of finally block is to maintain cleanup code.

Note : The speciality of finally block is it will be executed always irrespective of whether the exception raised or not raised and whether handled or not handled.

return Vs finally:

------------------

Even though return statement present in try or catch blocks first finally will be executed and after that only return statement will be considered. i.efinally block dominates return statement.

If return statement present try, catch and finally blocks then finally block return statement will be considered.

finally vs System.exit(0):

==========================

There is only one situation where the finally block won't be executed is whenever we are using System.exit(0) method.

When ever we are using System.exit(0) then JVM itself will be shutdown , in this case finally block won't be executed.

i.e., System.exit(0) dominates finally block.

Difference between final, finally, and finalize:

================================================

final:

 final is the modifier applicable for classes, methods and variables.

 If a class declared as the final then child class creation is not possible.

 If a method declared as the final then overriding of that method is not possible.

 If a variable declared as the final then reassignment is not possible.

finally:

 finally is the block always associated with try-catch to maintain clean up code which should be executed always irrespective of whether exception raised or not raised and whether handled or not handled.

finalize:

 finalize is a method, always invoked by Garbage Collector just before destroying an object to perform cleanup activities.

Note:

1. finally block meant for cleanup activities related to try block where as finalize() method meant for cleanup activities related to object.

2. To maintain clean up code finally block is recommended over finalize() method because we can't expect exact behavior of GC.

3.if we are not entering into the try block then the finally block won't be executed. Once we entered into the try block without executing finally block we can't come out.

4. We can take try-catch inside try i.e., nested try-catch is possible

5. The most specific exceptions can be handled by using inner try-catch and generalized exceptions can be handle by using outer try-catch.

Note: Default exception handler can handle only one exception at a time and that is the most recently raised exception.

Various possible combinations of try catch finally:

-----------------------------------------------------

1. Whenever we are writing try block compulsory we should write either catch or finally. i.e., try without catch or finally is invalid

2. Whenever we are writing catch block compulsory we should write try.

i.e., catch without try is invalid.

3. Whenever we are writing finally block compulsory we should write try.

i.e., finally without try is invalid.

4. In try-catch-finally order is important.

5. With in the try-catch -finally blocks we can take try-catch-finally.

i.e., nesting of try-catch-finally is possible.

6. For try-catch-finally blocks curly braces are mandatory.

throw Statement :

-----------------

Sometimes we need to create exception object explicitly and handover to the JVM manually by using throw keyword.

In general we can used throw keyword for custiomised exception not for pre defined exception.

After throw statement we can't take any statement directly otherwise we will get compile time error saying unreachable statement.

We can use throw keyword only for Throwable types otherwise we will get compile time error saying incomputable types.

throws statement :

------------------

In our program if there is any chance of raising checked exception then compulsory we should handle either by try catch or by throws keyword otherwise the code won't compile.

We can use throws keyword to delegate the responsibility of exception handling to the caller method. Then caller method is responsible to handle that exception.

Note :

 Hence the main objective of "throws" keyword is to delegate the responsibility of exception handling to the caller method.

 "throws" keyword required only checked exceptions. Usage of throws for unchecked exception there is no use.

 "throws" keyword required only to convince complier. Usage of throws keyword doesn't prevent abnormal termination of the program. Hence recommended to use try-catch over throws keyword.

we can use throws keyword only for Throwable types otherwise we will get compile time error saying incompatible types.

In our program with in the try block, if there is no chance of rising an exception then we can't right catch block for that exception otherwise we will get compile time error sayingexception XXX is never thrown in body of corresponding try statement. But this rule is applicable only for fully checked exception.

We can use throws keyword only for constructors and methods but not for classes.

Note: It is highly recommended to maintain our customized exceptions as unchecked by extending RuntimeException. We can catch any Throwable type including Errors also.

Version Enhancements:

---------------------

1.try with resource

---------------------

The main advantage of "try with resources" is the resources which are opened as part of try block will be closed automatically Once the control reaches end of the try block either normally or abnormally and hence we are not required to close explicitly so that the complexity of programming will be reduced.It is not required to write finally block explicitly and hence length of the code will be reduced and readability will be improved.

1. We can declare any no of resources but all these resources should be seperated with ;(semicolon)

try(R1 ; R2 ; R3)

{

-------------

-------------

}

2. All resources should be AutoCloseable resources. A resource is said to be auto closable if and only if the corresponding class implements the java.lang.AutoCloseable interface either directly or indirectly. All database related, network related and file io related resources already implemented AutoCloseable interface. Being a programmer we should aware and we are not required to do anything extra.

3. All resource reference variables are implicitly final and hence we can't perform reassignment with in the try block.

try(BufferedReader br=new BufferedReader(new FileReader("abc.txt"))) ;

{

br=new BufferedReader(new FileReader("abc.txt"));

}

output :

CE : Can't reassign a value to final variable br

4.Untill 1.6 version try should be followed by either catch or finally but 1.7 version we can take only try with resource without catch or finally

5.The main advantage of "try with resources" is finally block will become dummy because we are not required to close resources of explicitly.

2.Multi catch block

-------------------

Until 1.6 version ,Eventhough Multiple Exceptions having same handling code we have to write a separate catch block for every exceptions, it increases length of the code and reviews readability

try{

-----------------

-----------------

}

catch(ArithmeticException e) {

e.printStackTrace();

}

catch(NullPointerException e) {

e.printStackTrace();

}

catch(ClassCastException e) {

System.out.println(e.getMessage());

}

catch(IOException e) {

System.out.println(e.getMessage());

}

To overcome this problem Sun People introduced "Multi catch block" concept in 1.7 version.

The main advantage of multi catch block is we can write a single catch block , which can handle multiple different exceptions

try{

-----------------

-----------------

}

catch(ArithmeticException | NullPointerException e) {

e.printStackTrace();

}

catch(ClassCastException | IOException e) {

System.out.println(e.getMessage());

}

In multi catch block, there should not be any relation between Exception types(either child to parent Or parent to child Or same type , otherwise we will get Compile time error )

=========================================================================================================================================================================================

JAVA.LANG PACKAGE

-----------------

The FOllowing are the important classes present in the java.lang package

1. object class

2. String class

3. String BUffer

4. String Builder

5. Wrapper Classes

6. AutoBoxing and Auto Unboxing

-> For Writing any java program the most commonly required interfaces and classes are encapsulated in the separate package which is nothing but java.lang package.

-> By default this package is available to every java program.

Java.lang.object:

-----------------

- For any java object whether it is predefined or customised classes the most commonly required methods are encapsulated in a separate class which is nothing but object classe.

An object class acts as a root or parent or super for all the java classes so by defult

which ever the methods are available in the object class those methods are also available to the child class.

NOte : if our class does not extends any other class then our class is the direct child class of object.if it is extended any other class then our class is indirect child class of object class.

Common methods which are available in Object class:

---------------------------------------------------

1. toString()

2. hashCode()

3. equals(object o)

4. clone()

5. getClass()

6. finalize()

7. wait()

8. wait()

9. wait(long ms,long ns)

10.notify()

11.notifyAll()

to String():

------------

- we use this method to get the string representation of an object.

- whenever we are trying to print the object referenec internally toString() method will be executed.

- if our class doesnot contain any toString() method then object class toString() will be executed.

toString() of object class representation:

------------------------------------------

public String toString(){

return getClass().getName()+"@"+Integer.toHexaString(hashCode());

}

Note : to provide our own representation we have to override toString() in our class.

toString() : In String class, StringBuffer, StringBuilder, wrapper classes and in all collection classes toString() method is overridden for meaningful string representation.

Hence in our classes also highly recommended to override toString() method.

hashCode() :

------------

1.For every object jvm will generate a unique number which is nothing but hashCode.

2. Jvm will using hashCode while saving objects into hashing related data structures like HashSet, HashMap, and Hashtable etc.

3. If the objects are stored according to hashCode searching will become very efficient (The most powerful search algorithm is hashing which will work based on hashCode).

4. If we didn't override hashCode() method then Object class hashCode() method will be executed which generates hashCode based on address of the object but it doesn't mean hashCode represents address of the object.

5. Based on our programming requirement we can override hashCode() method to generate our own hashcode.

6. Overriding hashCode() method is said to be proper if and only if for every object we have to generate a unique number as hashcode for every object.

Note : If a class doesnot have toString() when we print object reference it call class toString() called which internally calls objectclass hashCode().

||y , if we override hashcode method in our class but not overriding the toString().in this case object class toString() method internally it call overriding hashCode().

Note :

1. if we are giving opportunity to Object class toString() method it internally calls

hashCode() method. But if we are overriding toString() method it may not call

hashCode() method.

2. We can use toString() method while printing object references and we can use

hashCode() method while saving objects into HashSet or Hashtable or HashMap.

equlas() :

----------

1. We can use this method to check equivalence of two objects.

2. If our class doesn't contain .equals() method then object class .equals() method will be executed which is always meant for reference comparison[address comparison]. i.e., if two references pointing to the same object then only .equals() method returns true .

3. In the above program Object class .equals() method got executed which is always meant for reference comparison that is if two references pointing to the same object then only .equals(() method returns true.In object class .equals() method is implemented as follows which is meant for reference comparison.

public boolean equals(Object o){

return(this==o);

}

When ever we are overriding .equals() method we have to consider the following things :

1. Meaning of content comparison i.e., whether we have to check the names are equal (or) roll numbers (or) both are equal.

2. If we are passing different type of objects (heterogeneous object) our.equals() method should return false but not ClassCastException i.e., we have to handle ClassCastException to return false.

3. If we are passing null argument our .equals() method should return false but not NullPointerException i.e., we have to handle NullPointerException to return false.

4. The following is the proper way of overriding .equals() method for content comparison in Student class.

\*\*\*In String class , Wrapper classes and all collection classes .equals( ) method is overriden for content comparision

\*\*\* StringBuffer and tringBuilder Classes .equals() is not for content comparison and it is only for reference comparison.

(==) is for both primitives and object comparison,if we compare with two heterogenous objects we will get incompatible compile time error.

.equals() is for only objects ,if we compare with heterogenous objects we will get false.

NOTE: in general we can use == (double equal operator) forreference comparison whereas .equals() method for content comparison.

in all wrapper classes, in string class, in all collection classes .equals() method is overridden for content comparison in our classes also it is highly recommended to override .equals() method.

getClass(): This method returns runtime class definition of an object.

finalize( ) : Just before destroying an object GC calls finalize( ) method to perform CleanUp activities.

wait( ) , notify( ) , notifyAll( ) : We can use these methods for inter thread communication.

STRING CLASS :

--------------

Immutability : Once we create a String object we can't perform any changes in the existing object. If we are try to perform any changes with those changes a new object will be created. This behavior is called immutability of the String object.

Mutable : Once we create an object. on that object we can perform any changes and this concept is nothing but mutability.

\*\* In String class .equals() is used for content comparison.

\*\* In StringBuffer and StringBUilder .equals is not overridden for content it uses object class toString method for reference comparison.

\*\*\*

String s=new String("bhaskar");

In this case two objects will be created one is on the heap the other one is in SCP(String constant pool) and s is always pointing to heap object.

\*\*\*

String s="bhaskar";

In this case only one object will be created in SCP and s is always referring that object in SCP.

NOTE :

1.Object creation in SCP is always optional 1st JVM will check is any object already created with required content or not. If it is already available then it will reuse existing object instead of creating new object. If it is not already there then only a new object will be created. Hence there is no chance of existing 2 objects with same content on SCP that is duplicate objects are not allowed in SCP.

2. Garbage collector can't access SCP area hence even though object doesn't have any reference still that object is not eligible for GC if it is present in SCP.

3. All SCP objects will be destroyed at the time of JVM shutdown automatically.

Example 1:

String s1=new String("bhaskar");

String s2=new String("bhaskar");

String s3="bhaskar";

String s4="bhaskar";

Note :

When ever we are using new operator compulsory a new object will be created on the Heap . There may be a chance of existing two objects with same content on the heap but there is no chance of existing two objects with same content on SCP . i.e., duplicate objects possible in the heap but not in SCP .

For every String Constant(Literal) one object will be created in SCP. Because of runtime operation if an object is required to create compulsory that object should be placed(created) on the heap but not SCP.

Importance Of String Constant Pool :

------------------------------------

1. In our program if any String object is required to use repeatedly then it is not

recommended to create multiple object with same content it reduces

performance of the system and effects memory utilization.

2. We can create only one copy and we can reuse the same object for every

requirement. This approach improves performance and memory utilization we

can achieve this by using "scp".

3. In SCP several references pointing to same object the main disadvantage in this

approach is by using one reference if we are performing any change the

remaining references will be impacted. To overcome this problem sun people

implemented immutability concept for String objects.

4. According to this once we creates a String object we can't perform any changes

in the existing object if we are trying to perform any changes with those changes

a new String object will be created hence immutability is the main disadvantage

of scp.

\*\*\*Note :

Because runtime operation if there is a change in content with those changes a new object will be created only on the heap but not in SCP.

\*\*\*If there is no change in content no new object will be created the same object will be reused.

Example 2:

class StringDemo {

public static void main(String[] args) {

String s1="bhaskar";

String s2=s1.toString();

System.out.println(s1==s2);//true

}

}

This rule is same whether object present on the Heap or SCP

\*\* String and Wrapper classes are immutable in java.

Interning of String Object:

---------------------------

by using heap reference,if we want to get corresponding CSP object then we should go for intern().if the corresponding object is not there in scp then intern() itself create that object and return it.

eg:1

class StringDemo {

public static void main(String[] args) {

String s1=new String("bhaskar");

String s2=s1.intern();

System.out.println(s1==s2); //false

String s3="bhaskar";

System.out.println(s2==s3);//true

}

Example 2:

class StringDemo {

public static void main(String[] args) {

String s1=new String("bhaskar");

String s2=s1.concat("software");

String s3=s2.intern();

String s4="bhaskarsoftware";

System.out.println(s3==s4);//true

}

}

after interning the object reference is pointing to SCP area so if we perform any == operation with literals of the same content then it will return true

String Class Constructor :

--------------------------

String s = new String();

String s = new String(String Literal);// craete equalent object on the heap.

String s = new String(StringBUffer);

String s = new String(char[] ch);

Imp Methods Of String Class:

----------------------------

1.charAt(index);

2.concat("String");

3.equals(object);

4.equalsIgnore(object);

5.subString(int begin);

6.subString(int begin,int end);

7.length()

8.replace(old char,new Char);

9.toLowerCase()

10.toUpperCase()

11.trim() //to remove spaces at the start and at the end but not in the middle.

12.indexOf(char) //first occurance

13.lastIndexOf(char) //last occurance.

Final vs immutability :

-----------------------

1. final modifier applicable for variables where as immutability concept applicable

for objects

2. If reference variable declared as final then we can't perform reassignment for

the reference variable it doesn't mean we can't perform any change in that

object.

3. That is by declaring a reference variable as final we won't get any immutability

nature .

4. final and immutability both are different concepts .

StringBuffer:

-------------

1. If the content will change frequently then never recommended to go for String

object because for every change a new object will be created internally.

2. To handle this type of requirement we should go for StringBuffer concept.

3. The main advantage of StringBuffer over String is, all required changes will be

performed in the existing object only instead of creating new object.(won't create

new object)

constructors

----------------

StringBuffer sb=new StringBuffer();

Creates an empty StringBuffer object with default initialcapacity "16".

Once StringBuffer object reaches its maximum capacity a new StringBuffer

object will be created with

Newcapacity=(currentcapacity+1)\*2.

StringBuffer sb=new StringBuffer(int initialcapacity);

Creates an empty StringBuffer object with the specified initial capacity

StringBuffer sb=new StringBuffer(String s);

Creates an equivalent StringBuffer object for the given String with

capacity=s.length()+16;

Important methods :

---------------------

1.length()

2.capacity()

3.charAt(index)

4.setCharAt(index,char) // to replace the character at the specified index with the provided character.

5.append() -- overloaded

// the type of args the append method takes : String,int,long,boolean,double,float,(index,object)

6.insert(index,overloaded types)

7.delete(int begin,int end)

8.deleteCharAt(index)

9.reverse()

10.setLength(length)// consider only specific number of character all other characters can be removed.

11.trimToSize() : // To deallocate the extra allocated free memory such that capacity and size are equal.

12.ensureCapacity(initialCapacity)// To increase the capacity dynamically(fly) based on our requirement.

Note :

Every method present in StringBuffer is syncronized hence at a time only one thread is

allowed to operate on StringBuffer object , it increases waiting time of the threads and

creates performence problems , to overcome this problem we should go for

StringBuilder.

StringBuilder (1.5v)

--------------------

1. Every method present in StringBuffer is declared as synchronized hence at a

time only one thread is allowed to operate on the StringBuffer object due to this,

waiting time of the threads will be increased and effects performance of the

system.

2. To overcome this problem sun people introduced StringBuilder concept in 1.5v.

StringBuilder is exactly same as StringBuffer(includung constructors and methods )

except the following differences :

String vs StringBuffer vs StringBuilder :

------------------------------------------

1. If the content is fixed and won't change frequently then we should go for String.

2. If the content will change frequently but Thread safety is required then we

should go for StringBuffer.

3. If the content will change frequently and Thread safety is not required then we

should go for StringBuilder.

Method chaining:

-----------------

1. For most of the methods in String, StringBuffer and StringBuilder the return

type is same type only. Hence after applying method on the result we can call

another method which forms method chaining.

2. Example:

3. sb.m1().m2().m3()...............

4. In method chaining all methods will be evaluated from left to right

===========================================================================================

WRAPPER CLASSES :

-----------------

1.To wrap primitives into object form so that we can handle primitives also just like objects.

2. To define several utility functions which are required for the primitives

1. All most all wrapper classes define the following 2 constructors one can take corresponding primitive as argument and the other can take String as argument.

-> If the String is not properly formatted i.e., if it is not representing number then

we will get runtime exception saying "NumberFormatException

2.Float class defines 3 constructors with float, String and double arguments.

3.Character class defines only one constructor which can take char primitive as argument there is no String argument constructor.

4.Boolean class defines 2 constructors with boolean primitive and String arguments. If we want to pass boolean primitive the only allowed values are true, false where case should be lower case.If we are passing String argument then case is not important and content is not important. If the content is case insensitive String of true then it is treated as true in all other cases it is treated as false

NOTE :

1) In all wrapper classes toString() method is overridden to return its content.

2) In all wrapper classes .equals() method is overridden for content compression

Utility methods :

-----------------

1. valueOf() method :We can use valueOf() method to create wrapper object for the given primitive or String this method is alternative to constructor.

Every wrapper class except Character class contains a static valueOf() method to create wrapper object for the given String.

2. XXXValue() method: We can use xxxValue() methods to convert wrapper object to primitive.

Every number type wrapper class (Byte, Short, Integer, Long, Float, Double) contains

the following 6 xxxValue() methods to convert wrapper object to primitives.

1)public byte byteValue()

2)public short shortValue()

3)public int intValue()

4)public long longValue()

5)public float floatValue()

6)public double doubleValue();

charValue() method:

-------------------

Character class contains charValue() method to convert Character object to char

primitive.

public char charValue();

Example:

class WrapperClassDemo {

public static void main(String[] args) {

Character ch=new Character('a');

char c=ch.charValue();

System.out.println(c);//a

}

}

booleanValue() method:

-----------------------

Boolean class contains booleanValue() method to convert Boolean object to boolean

primitive.

public boolean booleanValue( );

Example:

class WrapperClassDemo {

public static void main(String[] args) {

Boolean b=new Boolean("ashok");

boolean b1=b.booleanValue();

System.out.println(b1);//false

}

}

3. parseXxx() method: We can use this method to convert String to corresponding primitive.

Every wrapper class except Character class contains a static parseXxx() method to

convert String to corresponding primitive.

public static primitive parseXxx(String s);

Example:

class WrapperClassDemo {

public static void main(String[] args) {

int i=Integer.parseInt("10");

boolean b=Boolean.parseBoolean("ashok");

double d=Double.parseDouble("10.5");

System.out.println(i);//10

}

}

4. toString() method: We can use toString() method to convert wrapper object (or) primitive to String

public String toString();

1. Every wrapper class (including Character class) contains the above toString()

method to convert wrapper object to String.

2. It is the overriding version of Object class toString() method.

3. Whenever we are trying to print wrapper object reference internally this

toString() method only executed

Every wrapper class contains a static toString() method to convert primitive to String.

public static String toString(primitive p);

case 1: (String/primitive)---> valueOf()--->WrapperObject

case 2: (Wrapper Object)----> xxxValue()---->Primitive values

case 3: (String)-----> ParseXXX-------> primitive

case 4: (object/primitive)------> toString()/toSTring()---->string form.

1.String, StringBuffer, StringBuilder and all wrapper classes are final classes.

2. The wrapper classes which are not child class of Number are Boolean and Character.

3. The wrapper classes which are not direct class of Object are Byte, Short, Integer,Long, Float, Double.

4. Sometimes we can consider Void is also as wrapper class.

5. In addition to String objects , all wrapper class objects also immutable in java.

AUTOBOXING and AUTO UNBOXING:

-----------------------------

Until 1.4 version we can't provide wrapper object in the place of primitive and primitive

in the place of wrapper object all the required conversions should be performed

explicitly by the programmer

Autoboxing : Automatic conversion of primitive to wrapper object by compiler is called Autoboxing by valueOf()

AutoUnboxing : automatic conversion of wrapper object to primitive by compiler is called Autounboxing. by XXXValue().

If we provide null reference for autounboxing , we will get NullPointerException

Note :

->All wrapper objects are immutable that is once we created a wrapper object we can't perform any changes in the existing object.

->If we are trying to perform any changes with those changes a new object will be created.

BUffer Concept in wrapper classes just like String Constant Pool

Conclusions :

1. To implement the Autoboxing concept in every wrapper class a buffer of objects

will be created at the time of class loading.

2. By Autoboxing if an object is required to create 1st JVM will check whether that

object is available in the buffer or not.

3. If it is available then JVM will reuse that buffered object instead of creating new

object.

4. If the object is not available in the buffer then only a new object will be created.

This approach improves performance and memory utilization

But this buffer concept is available only in the following cases :

Byte Always

Short -128 To 127

Integer -128 To 127

Long -128 To 127

Character 0 To 127

Boolean Always

eg: Integer i1= new Integer(10);

Integer i2= new Integer(20);

System.out.println(i1==i2);//false

eg: Integer i1= 10; //internally it create with Interger.valueOf(10);

Integer i2= new Integer(20);

System.out.println(i1==i2);//false

eg: Integer i1= Interger.valueOf(10);

Integer i2=10;

System.out.println(i1==i2);//true

Integer i1= 10;

Integer i2=10;

System.out.println(i1==i2);//true

Integer i1= Interger.valueOf(10);

Integer i2= Interger.valueOf(10);

System.out.println(i1==i2);//true

Note:

When compared with constructors it is recommended to use valueOf() method to create

wrapper object.

Overloading with respect to widening, Autoboxing and var-arg methods :

----------------------------------------------------------------------

-> Widening dominates Autoboxing

-> Widening vs var-arg method

-> Autoboxing dominates var-arg method.

Note : While resolving overloaded methods compiler will always gives the

presidence in the following order :

1. Widening

2. Autoboxing

3. Var-arg method.

AutoBoxing followed by widening is possible in java but in converse Widening followed by autoboxing is not possible

in widening there should be some relationship exist.

Which of the following declarations are valid ?

1. int i=10 ; //valid

2. Integer I=10 ; //valid

3. int i=10L ; //invalid CE :

4. Long l = 10L ; // valid

5. Long l = 10 ; // invalid CE :

6. long l = 10 ; //valid

7. Object o=10 ; //valid (autoboxing followed by widening)

8. double d=10 ; //valid

9. Double d=10 ; //invalid

10. Number n=10; //valid (autoboxing followed by widening