***Main Method***

Whether class contains main method or not and whether main method is declared according to requirement or not these things will not be checked by compiler.

At runtime JVM is responsible to check these things. If JVM is unable to find main method then we will get runtime exception .

**EG:**

Class Test

{

}

Output: Main method not found in class Rough, please define the main method as:public static void main(String[] args)

At runtime JVM always searches for main method with the following prototype ,

Public static void main(String args[])

Public : To call by JVM from anywhere.

Static : Without existing any object JVM has to call this method.

Void : main method will not return anything to JVM.

Main : it is the name configured inside JVM.

String args[] : Command line arguments.

The above syntax is very strict and if we perform any change then we will get runtime exception as Main method not found in class Rough, please define the main method as:public static void main(String[] args)

Even though it is very strict the following changes are acceptable :

1. Instead of public static we can take static public i.e order of modifiers is not important .
2. We can declare string array in any acceptable form.

main(String[] args)

main(String []args)

main(String args[])

1. Instead or args we can take any valid java identifier.

main(String[] darsh)

1. We can replace string array with var-arg parameter

main(String… args)

1. We can declare main method with the following modifiers

1.final

2.synchronized

3.strictfp

EG:**public** **class** Rough {

**static** **final** **synchronized** **strictfp** **public** **void** main(String… args) {

System.***out***.print("Valid main method");

}

}

**Which of the following main method declarations are valid?**

1. public static void main(String args)

Invalid

1. public static void Main(String[] args)

Invalid

1. public void main(String[] args)

Invalid

1. public static int main(String[] args)

Invalid

1. final synchronized strictfp public void main(String[] args)

Invalid

1. final synchronized strictfp public static void main(String[] args)

Valid

1. public static void main(String… args)

Valid

**Which of the following gives compile time error?**

* None gives compile time. All the above examples compile successfully but except 6th and 7th all give declaration give runtime error as noSuchMethodError as main.

**Case 1.**

Overloading of the main method is possible but JVM will always call String[] argument main method only. The other overloaded method should be called explicitly like normal method call.

EG:

**public** **class** Rough {

**static** **public** **void** main(String[] args) {

System.***out***.print("String[]");

}

**public** **static** **void** main(**int**[] args)

{

System.***out***.print("int[]");

}

}

**Output: String[]**

**Case 2.**

Inheritance concept is applicable for main method . hence while executing child class if child does not contain main method then parent class main method will be executed.

We will not get any runtime error if main method is not present in child class.

EG:

**public** **class** Rough {

**static** **public** **void** main(String[] args) {

System.***out***.print("Parent main");

}

**class** r **extends** Rough{

}

**Case 3.**

**public** **class** Rough {

**static** **public** **void** main(String[] args) {

System.***out***.print("Parent main");

}

**class** r **extends** Rough

{

**public** **static** **void** main(String[] args)

{

System.***out***.print("Child main");

}

}

Output: Parent main

It seems overriding concept is applicable for main method but it is not overriding and it is method hiding.

**Note:**

For main method inheritance and overloading concepts are applicable but overriding concept is not applicable.

Instead of overriding method hiding is applicable.

1.7 version enhancements w.r.t main method

* Until 1.6 version if the class does not contain main method then we will get runtime exception saying noSuchMethodError:main

But from 1.7 version onwards instead of noSuchMethodrror we will get more elobarated error information as: Main method not found in class Rough, please define the main method as:public static void main(String[] args)

* From 1.7 version onwards main method is mandatory to start program execution . Hence even though class contains static block it will not be executed if the class doesn’t contain main method:

EG:

Class Test

{

Static

{

Sop(“Static block”);

}

}

1.6 version o/p: Static block

NoSuchMethodError:main

1.7 version o/p:

Main method not found in class rough, please define the main method as :public static void main(String[] args)

* Class Test

{

Static

{

Sop(“Static block”);

System.exit(0);

}

}

1.6 version o/p:

Static Block

1.7 version o/p:

Main method not found in class rough, please define the main method as :public static void main(String[] args)

* Class Test

{

Static

{

Sop(“Static block”);

}

Public static void main(String[] args)

{

Sop(“main method”);

}

}

1.6 version o/p:

Static Block

Main method

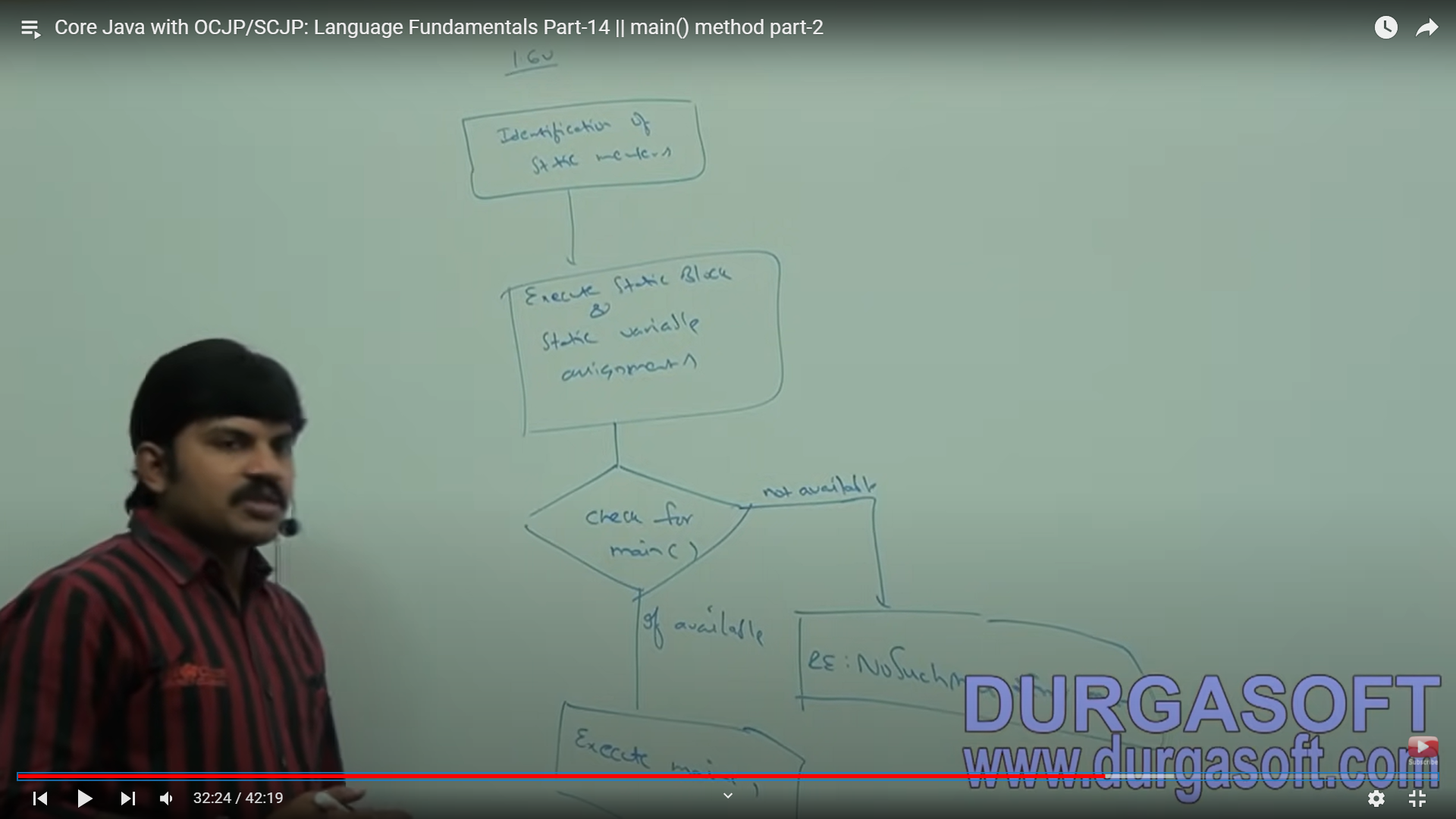
1.7 version o/p:

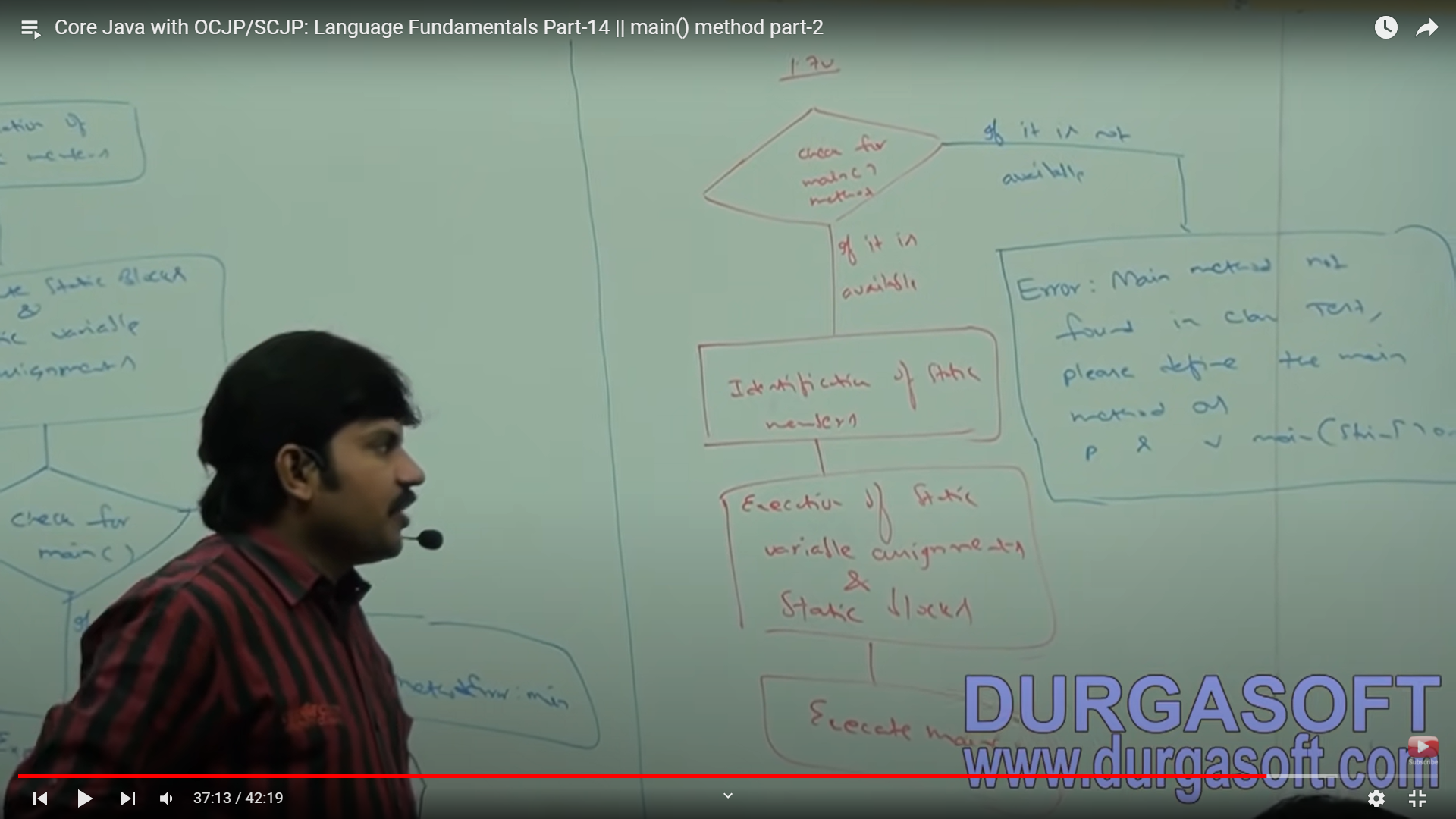
Static block

Main method

**Without writing main method is it possible to print some statements to the console?**

Yes, by using static block. But this rule is applicable until 1.6 version. From 1.7 version onwards it is impossible to print statements to the console without writing main method.





***Command Line Arguments***

The arguments which are passed from command prompt are called command line arguments. With these command line arguments JVM will create an array and by passing that array as arguments JVM will call main method.

The main objective of command line arguments is we can customize behaviour of the main method.

**EG 1:**

**public** **class** Rough {

**static** **public** **void** main(String[] args) {

System.***out***.print("Parent main");

**for**(**int** i=0;i<=args.length;i++)

{

System.***out***.print(args[i]);

}

}

}

Output:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 0 out of bounds for length 0 at Rough.main(Rough.java:6)

**EG 2:**

**public** **class** Rough {

**static** **public** **void** main(String[] args) {

String[] argh = {"x", "y", "z"};

args = argh;

**for**(String s:args)

{

System.***out***.print(s);

}

}

}

Output: xyz

Within main method command line arguments are available in string form.

**EG 3:**

**public** **class** Rough {

**static** **public** **void** main(String[] args) {

System.***out***.print(args[0]+args[1]);

}

}

Input = java Rough 10 20

Output: 1020

**EG 4:**

Usually space itself is a separator between command line arguments. If our command line argument itself contains space we have to enclose that command line argument within double quotes.

**public** **class** Rough {

**static** **public** **void** main(String[] args) {

System.***out***.print(args[0]);

}

}

Input = “note book”

Output: note book

**Java Coding Standards**

Whenever we are writing java code it is highly recommended to follow coding standards. Whenever we are writing any component its name should reflect the purpose of that component.

The main advantage of this approach is readability and maintainability of the code is improved.

1. class A

{

Public int m1(int x, int y)

{

return x+y;

}

}

1. package com.durgasoft.scjp;

public class calculator

{

public static int add(int number1, int number2)

{

return number1+number2;

}

}

* ***Coding standards for classes***

Usually class names are nouns and should start with uppercase and if contains multiple words every inner word should start with uppercase character.

EG :

1. String
2. StringBuffer
3. Account
4. Dog

* ***Coding standards for interfaces***

Usually interface names are adjectives and should start with uppercase character and if it contains multiple word every inner word should start with uppercase character.

EG :

1. Runnable
2. Serializable
3. Comparable

* ***Coding standards for methods***

Usually method names are either verbs or verb-noun combination. Should start with lowercase alphabet symbol and if it contains multiple words then every inner word should start with uppercase character i.e camelCase convention

EG :

1. print()
2. sleep()
3. run()
4. eat()
5. start()
6. getName()
7. setSalary()

* ***Coding standards for variables***

Usually variable names are nouns. Should starts with lowercase alphabet symbol and if it contains multiple words then it should follow camelCase convention.

EG :

1. name
2. age
3. salary
4. mobileNumber

* ***Coding standards for Constants***

Usually constant names are nouns which should contain only uppercase characters an dif it contains multiple words then these words are separated with underscore symbol.

Usually we can declare constants with public static and final modifiers.

EG :

1. MAX\_VALUE
2. MAX\_PRIORITY
3. NORM\_PRIORITY
4. MIN\_PRIORITY
5. PI

* ***Javabean Coding standards***

A javabean is a simple java class with private properties and public getter and setter methods.

EG :

**public** **class** StudentBean

{

**private** String name;

**public** **void** setName(String name)

{

**this**.name = name;

}

**public** String setName()

{

**return** name;

}

}

Class name should end with ‘Bean’ is not an official compulsion from SUN.

*Syntax for setter method:*

1. It should be public method
2. The return type should be void
3. Method name should be prefixed with set
4. It should take some argument i.e it should not be no argument method

*Syntax for getter method:*

1. It should be public method
2. The return type should not be void
3. Method name should prefixed with get
4. It should not take any argument.

**NOTE :**

For Boolean properties getter method name can be prefixed with either ‘get’ or ‘is’. But recommended to use ‘is’.

1. public Boolean getEmpty()

{

return empty;

}

1. public Boolean isEmpty()

{

return empty;

}

Both are valid but the 2nd is recommended.

* ***Coding standards for Listeners***

**Case 1 :**

Method name should be prefixed with add.

EG :

1. public void addMyActionListener(MyActionListener l) ..valid
2. public void registerMyActionListener(MyActionListener l) ..invalid
3. public void addMyActionListener(ActionListener l) ..invalid

**Case 2 :**

Method name should be prefixed with remove.

EG :

1. public void removeMyActionListener(MyActionListener l) ..valid
2. public void unRegisterMyActionListener(MyActionListener) ..invalid
3. public void removeMyActionListener(ActionListener l) ..invalid
4. public void deleteMyActionListener(MyActionListener l) ..invalid