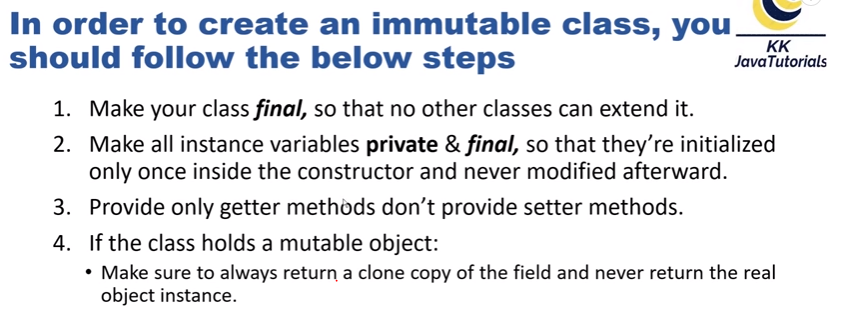
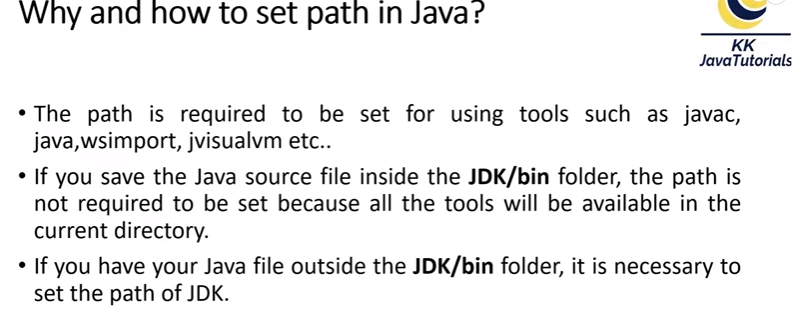
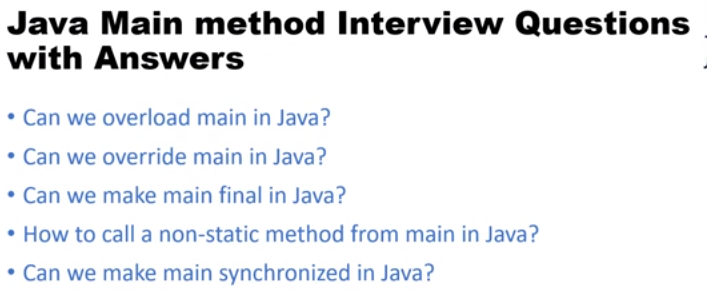
|  |
| --- |
| **public** **class** javaTrickOveloding {  **public** **static** **void** main(String args[]) {  *test*(**null**);  }  **public** **static** **void** test(Object obj) {  System.***out***.println("object impl called");  }    /\*public static void test(String obj) {  System.out.println("String impl called");  }\*/    **public** **static** **void** test(Integer obj) {  System.***out***.println("integer impl called");  }  }  Output:  integer impl called |

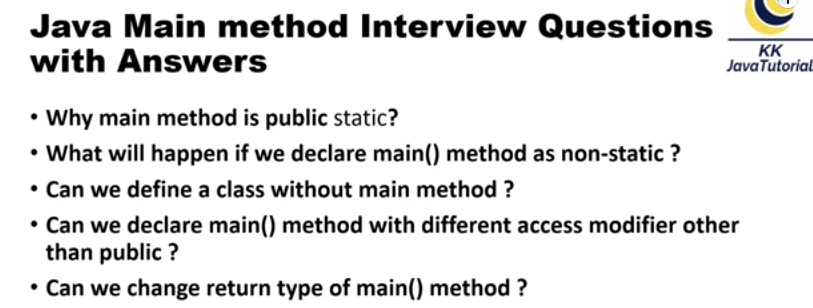


|  |
| --- |
| **public** **class** Student {  // by default string and all wraper classes are immutable  **private** **final** String name;  **private** **final** Integer id;  **private** **final** List<String> subjects;  **private** **final** Address address;    **public** Student(String name, Integer id, List<String> subjects, Address address) {  **this**.name = name;  **this**.id = id;  **this**.subjects = subjects;  **this**.address = address;  }  **public** String getName() {  **return** name;  }  **public** Integer getId() {  **return** id;  }  **public** List<String> getSubjects() {  **return** **new** ArrayList(subjects);  }  **public** Address getAddress() {  Address obj = **new** Address();  obj.setCity(address.getCity());  obj.setPinCode(address.getPinCode());  obj.setState(address.getState());  **return** obj;  }  }  **public** **class** Address {  **private** String city;  **private** String state;  **private** **int** pinCode;  **public** String getCity() {  **return** city;  }  **public** **void** setCity(String city) {  **this**.city = city;  }  **public** String getState() {  **return** state;  }  **public** **void** setState(String state) {  **this**.state = state;  }  **public** **int** getPinCode() {  **return** pinCode;  }  **public** **void** setPinCode(**int** pinCode) {  **this**.pinCode = pinCode;  }  @Override  **public** String toString() {  **return** "Address [city=" + city + ", state=" + state + ", pinCode=" + pinCode + "]";  }  }  Client Program:  **public** **class** Client {  **public** **static** **void** main(String[] args) {  // **TODO** Auto-generated method stub    List<String> subjects = **new** ArrayList();  subjects.add("java");  subjects.add("scala");    Address address = **new** Address();  address.setCity("KLD");  address.setPinCode(152);  address.setState("A");    Student s1 = **new** Student("nagendra", 101, subjects, address);  System.***out***.println("\*\*\*\*\*\*before modification\*\*\*\*\*\*");      System.***out***.println("name:"+ s1.getName());  System.***out***.println("id:"+ s1.getId());  System.***out***.println("subject count:"+ s1.getSubjects().size());  System.***out***.println("address:"+ s1.getAddress());    // modifyes address and subjects    s1.getAddress().setCity("ATP");  s1.getSubjects().add("jpa");    System.***out***.println("\*\*\*\*\*\*After modification\*\*\*\*\*\*");      System.***out***.println("name:"+ s1.getName());  System.***out***.println("id:"+ s1.getId());  System.***out***.println("subject count:"+ s1.getSubjects().size());  System.***out***.println("address:"+ s1.getAddress());  }  }  Output:  \*\*\*\*\*\*before modification\*\*\*\*\*\*  name:nagendra  id:101  subject count:2  address:Address [city=KLD, state=A, pinCode=152]  \*\*\*\*\*\*After modification\*\*\*\*\*\*  name:nagendra  id:101  subject count:2  address:Address [city=KLD, state=A, pinCode=152] NOTE: Do all properties of an Immutable Object need to be final in java? We know that String and Integer are by default immutable, is it require final modifier for this variable ?  Case 1: if we add final even though inside constructor also re assign not possible  Case 2 : if we remove final , inside constructor also re assign possible |





|  |
| --- |
| **First scenario:**  **public** **class** JavaMainMethodOverloding {  **public** **static** **void** main(String[] args) {  *main*("nag");  **int** result = *main*(20,10);  System.***out***.println(result);  }  **public** **static** **void** main(String name) {  System.***out***.println(name);  }    **public** **static** **int** main(**int** a, **int** b) {  **return** a+b;  }  }  Output:  nag  30  Second scenerio: can we override main method in java  Ans : No because static method can’t override,  Case 1: in this case compiler doesn’t show error  **public** **class** OverrideMainMethod {  **public** **static** **void** main(String[] args) {  System.***out***.println("main method parent");  }  }  **class** child **extends** OverrideMainMethod {  //@override  **public** **static** **void** main(String[] args) {  System.***out***.println("main method parent");  }  }  Case 2: if we add @override keyword in child method it will show error in compile time  Case 3: can we make main final in java  Ans: Yes we can make final main method, jvm has no issue.  **public** **class** OverrideMainMethod {  **public** **static** **final** **void** main(String[] args) {  System.***out***.println("main method parent");  }  }  /\*class child extends OverrideMainMethod {  public static void main(String[] args) {  System.out.println("main method parent");  }  }\*/  Case 4: how can call non-static method call from main method  Ans: every non static method call using objec  **public** **class** OverrideMainMethod {  **public** **static** **final** **void** main(String[] args) {  **new** OverrideMainMethod().*doSome*("demo ");  }  **public** **static** **void** doSome(String val) {  System.***out***.println(val);  }  }  Case 5: can we made main method synchronize in java  Ans: yes we can add no issue, any jvm call only once.  **public** **class** OverrideMainMethod {  **public** **static** **synchronized** **void** main(String[] args) {  System.***out***.println("hello");  }  }  **Output:**  hello |



Case 1: why main method is public static

Ans:

Since the main method is static Java virtual Machine can call it without creating any instance of a class which contains the main method.

Since C and C++ also have similar main method which serves as entry point for program execution, following that convention will only help Java.

If main method were not declared static than JVM has to create instance of main Class and since constructor can be overloaded and can have arguments with different. So jvm does’t not decided which method call

Anything which is declared in [class in Java](http://javarevisited.blogspot.com/2011/10/class-in-java-programming-general.html) comes under reference type and requires object to be created before using them but static method and static data are loaded into separate memory inside JVM called context which is created when a class is loaded. If main method is static than it will be loaded in JVM context and are available to execution.

**Why main method is public in Java**

Java specifies several access modifiers e.g. private, protected and public. Any method or variable which is declared public in Java can be accessible from outside of that class. Since the main method is public in

Java, JVM can easily access and execute it.

**Why the main method is void in Java**

Since the main method in Java is not supposed to return any value, it's made void which simply means main is not returning anything.

Case 2: what will happens main method is declared no static

Ans: in this case compilation successful but we will get runtime error

|  |
| --- |
| **public** **class** OverrideMainMethod {  **public** **void** main(String[] args) {  System.***out***.println("hello");  }  }  Output:  Error: Main method is not static in class com.mng.interview.OverrideMainMethod, please define the main method as:  public static void main(String[] args) |

Case 3: can we define class without main method

In this depending on jdk versions.

If java 4 to 6 versions we can define and run using static block.

If java 7 to above its not possible, compilation is successes full but will get runtime exception

|  |
| --- |
| **public** **class** OverrideMainMethod {  **static** {  System.***out***.println("main class");  }  }  Output:  Error: Main method not found in class com.mng.interview.OverrideMainMethod, please define the main method as:  public static void main(String[] args)  or a JavaFX application class must extend javafx.application.Application |

Case 4: can we declare main method different access modifier other than public:

Ans: no if use completion fine but we will get runtime error, method not found

Case 5: can we change the main method return type

Ans: No if we declare main method different return type other than void.

Compilation is fine but we will get run time error

|  |
| --- |
| **public** **class** OverrideMainMethod {  **public** **static** **int** main(String[] args) {  System.***out***.println("hello");  **return** 0;  }  }  **Output:**  Error: Main method must return a value of type void in class com.mng.interview.OverrideMainMethod, please  define the main method as:  public static void main(String[] args) |

Question: What is out of the program

|  |
| --- |
| **class** A {  **publicvoid** printData() {  System.***out***.println("A");  }  }  **class** B **extends** A{  **publicvoid** printData() {  System.***out***.println("B");  }  }  **class** C **extends**B{  **public** C() {  System.***out***.println("C");  }  }  **publicclass** DemoInterviewClass {  **publicstaticvoid** main(String[] args) {  C obj = **new** C();  }}  OutPut:  C |

Case 2

|  |
| --- |
| **class** A {  **public** A() {  System.***out***.println("A");  }  }  **class** B **extends** A{  **public** B() {  System.***out***.println("B");  }  }  **class** C **extends** B {  **public** C() {  System.***out***.println("C");  }  }  **publicclass** DemoInterviewClass {  **publicstaticvoid** main(String[] args) {  C obj = **new** C();  }}  Output:  A  B  C |

Case 3

|  |
| --- |
| **class** A {  **public** A() {  System.***out***.println("A");  }  }  **class** B **extends** A{  **public** B() {  System.***out***.println("B");  }  }  **class** C **extends** B {  **publicvoid** print() {  System.***out***.println("C");  }  }  **publicclass** DemoInterviewClass {  **publicstaticvoid** main(String[] args) {  C obj = **null**;  obj.print();  }}  Output: compilation is fine but we will get null pointer exception |

Case 4:

|  |
| --- |
| **class** SampleDemo {  SampleDemo obj = **new** SampleDemo();  }  **publicclass** javainterviewcase4 {  **publicstaticvoid** main(String[] args) {  SampleDemo obj = **new** SampleDemo();  }  }  Output:  Exception in thread "main" java.lang.StackOverflowError  at com.mng.java.SampleDemo.<init>(javainterviewcase4.java:5)  at com.mng.java.SampleDemo.<init>(javainterviewcase4.java:5) |

Case 5:

|  |
| --- |
| **publicclass** javainterviewcase4 {  **publicstaticvoid** main(String[] args) {  Stringstr = **new**String(**null**);  }}  Note: in this case compilation ambiguity error occurred because internally String class contains two constructors  Public String(String str)  {--  }  Public String(StringBuffer sb){  ---  }  Like  **class** SampleDemo {  **public** SampleDemo() {}  **public** SampleDemo(String str) {  System.***out***.println("String Constructor");  }  **public** SampleDemo(StringBuffer str) {  System.***out***.println("StringBuffer Constructor");  }  }  **publicclass** javainterviewcase4 {  **publicstaticvoid** main(String[] args) {  SampleDemo obj = **new** SampleDemo(**null**);  }  }  Above also ambiguity error |

Case 5:

|  |
| --- |
| **publicclass** javainterviewcase4 {  **publicstaticvoid** main(String[] args) {  Integer it = **new** Integer(**null**);  }}  In internally Interger class chcking null  Like  **if** (s == **null**) {  **thrownew** NumberFormatException("null");  }  Output:  Exception in thread "main" java.lang.NumberFormatException: null  at java.lang.Integer.parseInt(Integer.java:542)  at java.lang.Integer.<init>(Integer.java:867) |

Case 6: in super classes having parameterized constructor along with default constructor should be exists otherwise we compilation error.

|  |
| --- |
| **class** A {  **public** A() { System.***out***.println("A"); }  **public** A(**int** a) { System.***out***.println("parameter A");  }  }  **class** B **extends** A{  **public** B() { System.***out***.println("B");}  **public** B(**int** b) { System.***out***.println("parameter b"); }  **public** **void** printData() {  System.***out***.println("B");  }  }  **class** C **extends** B{  **public** C() {  System.***out***.println("C");  }  }  **public** **class** javaTrickOveloding {  **public** **static** **void** main(String[] args) {  C obj = **new** C();  }  }  Output:  A  B  C |

Case 7: static method calls without any issue even though the reference contains null;

|  |
| --- |
| **public** **class** javaTrickOveloding {  **public** **static** **void** main(String[] args) {  javaTrickOveloding obj = **null**;  obj.*display*();    // above statement equals to  // javaTrickOveloding.display();  }  **public** **static** **void** display() {  System.***out***.println("do some method");  }  }  Output:  do some method |

width minimum maximum

SIGNED

byte: 8 bit -128 +127

short: 16 bit -32 768 +32 767

int: 32 bit -2 147 483 648 +2 147 483 647

long: 64 bit -9 223 372 036 854 775 808 +9 223 372 036 854 775 807

UNSIGNED

char 16 bit 0 +65 535

Case 8:

|  |
| --- |
| **public** **class** javaTrickOveloding {  **public** **static** **void** main(String[] args) {  Integer i1 = 128;  Integer i2 = 128;  System.***out***.println(i1 == i2); // false    Integer i3 = 127;  Integer i4 = 127;  System.***out***.println(i3 == i4); //true  }  }  Output  False  True  **public** **class** javaTrickOveloding {  **public** **static** **void** main(String[] args) {  Integer i1 = 125;  Integer i2 = 125;  System.***out***.println(i1 == i2);    Integer i3 = 127;  Integer i4 = 127;  System.***out***.println(i3 == i4);  }  }  Output:  True  True |

Case 9: compilation time happen because print stream class contains overloaded println() methods

|  |
| --- |
| **public** **class** javaTrickOveloding {  **public** **static** **void** main(String[] args) {  System.***out***.println(**null**);    /\* in printstram class contans method oveloading  \* public void println(String str) {}  public void println(Char[] str) {}\*/  }  } |

Case 10: compilation happens don’t declare any static variable with inner class

|  |
| --- |
| **public** **class** javaTrickOveloding {  **public** **static** **void** main(String[] args) {  **public** **static** String str = "playload";  **final** **class** constantData {  **public** **void** printdata() {  System.***out***.println(str);  }  }  }  }  In this case no compilations issue  **public** **class** javaTrickOveloding {  **public** **static** String *str* = "playload";  **public** **static** **void** main(String[] args) {    **final** **class** constantData {  **public** **void** printdata() {  System.***out***.println(*str*);  }  }  }  } |

Case 11: what do the expression 1.0/0.0 will return ? will it throw exception? Any compile time?

public static final double POSITIVE\_INFINITY = 1.0 / 0.0;

public static final double NEGATIVE\_INFINITY = -1.0 / 0.0;

public static final double NaN = 0.0d / 0.0;

|  |
| --- |
| **public** **class** javaTrickOveloding {  **public** **static** **void** main(String[] args) {  Double d = 1.0/0.0;  System.***out***.println(d);      Double d2 = 0.0/0.0;  System.***out***.println(d2);  }  }  Infinity  NaN |

Case 12: find out output of the program

|  |
| --- |
| **class** parent{  **public** **static** **void** m1() {  System.***out***.println("parent m");  }  }  **class** child **extends** parent {  **public** **static** **void** m1() {  System.***out***.println("child m1");  }  }  **public** **class** DemoJava {  **public** **static** **void** main(String[] args) {  child c = **new** child();  c.*m1*();  parent p = **new** child();  p.*m1*();  }  }  Output:  child m1  parent m |

Case 13:

|  |
| --- |
| **class** Base {  // Static method in base class which will be hidden in subclass  **public** **static** **void** display() {  System.***out***.println("Static or class method from Base");  }  // Non-static method which will be overridden in derived class  **public** **void** print() {  System.***out***.println("Non-static or Instance method from Base");  }  }  // Subclass  **class** Derived **extends** Base {  // Static is removed here (Causes Compiler Error)  **public** **void** display() {  System.***out***.println("Non-static method from Derived");  }  // Static is added here (Causes Compiler Error)  **public** **static** **void** print() {  System.***out***.println("Static method from Derived");  }  } |