**CSE 310**

**Week 3**

1. **‘Final’ keyword**
   1. **Final variable:**
      1. **Syntax: final <variable type> <variable name>;**
      2. **If you** make any variable as final, you cannot change the value of final variable (It will be constant).
      3. We must initialize a final variable, otherwise compiler will throw compile-time error. A final variable can only be initialized once, either via an [initializer](https://www.geeksforgeeks.org/g-fact-26-the-initializer-block-in-java/) or an assignment statement.
      4. We can initialize a final variable when it is declared. This approach is the most common. A final variable is called **blank final variable**, if it is **not** initialized while declaration.
      5. A blank final variable can be initialized inside [**instance-initializer block**](https://www.geeksforgeeks.org/instance-initialization-block-iib-java/)**or inside constructor.** If you have more than one constructor in your class then it must be initialized in all of them, otherwise compile time error will be thrown.
      6. **A** blank final static variable **MUST** be initialized inside [static block](https://www.geeksforgeeks.org/g-fact-79/).
   2. **Reference Final variable:**
      1. Whena final variable is a reference to an object, then this final variable is called reference final variable.
      2. Asyou know that a final variable cannot be re-assign. But in case of a reference final variable, internal state of the object pointed by that reference variable can be changed.
   3. **Final local variable:**
      1. Whena final variable is created inside a method/constructor/block, it is called local final variable, and it must initialize once where it is created.

class A

{

    public static void main(String[] args)

    {

        int arr[] = {1, 2, 3};

        // final with for-each statement

        // legal statement

        for (final int i : arr)

            System.out.print(i + " ");

    }

}

* 1. **Final class:**
     1. **Syntax: final class <class name> { }**
     2. When a class is declared with final keyword, it is called a final class. A final class cannot be extended (inherited).
     3. **Reasons to use final class**
        + - One is definitely to prevent [inheritance](https://www.geeksforgeeks.org/inheritance-in-java/), as final classes cannot be extended. For example, all [Wrapper Classes](https://www.geeksforgeeks.org/wrapper-classes-java/) like [Integer](https://www.geeksforgeeks.org/java-lang-integer-class-java/),[Float](https://www.geeksforgeeks.org/java-lang-float-class-in-java/) etc. are final classes. We cannot extend them.
          - The other use of final with classes is to [create an immutable class](https://www.geeksforgeeks.org/create-immutable-class-java/) like the predefined [String](https://www.geeksforgeeks.org/string-class-in-java/) class. You cannot make a class immutable without making it final.
  2. **Final method:**
     1. **Syntax: <access modifier> final <return type> <method name> ( ) { }**
     2. When a method is declared with final keyword, it is called a final method. A final method cannot be [overridden](https://www.geeksforgeeks.org/overriding-in-java/).

1. **Abstract keyword:**
   1. **Abstraction:**
      1. **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
      2. **Another way,** it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.
      3. **There** are two ways to achieve abstraction in java
         * Abstract class (0 to 100%)
         * Interface (100%)
   2. **Abstract class:**
      1. **Syntax: <access modifier> abstract class <class name> { }**
      2. **An abstract class cannot be instantiated.**
      3. **An abstract class needs to be inherited.**
      4. The class that inherits abstract class **MUST** implement all of its abstract methods.
      5. Anabstract class can have data member, abstract method, method body, constructor and even main () method.
      6. The abstract class can also be used to provide some implementation of the interface. In such case, the end user may not be forced to implement all the methods of the interface.
      7. Please refer to codes.
   3. **Abstract method:**
      1. **Syntax: <access modifier> abstract <return type> <method name> ( ) { }**
      2. If there is any abstract method in a class, that class must be abstract.
2. **Interface:**
   1. **An interface in java** is a blueprint of a class. It has static constants and abstract methods.
   2. **The** interface in java is **a mechanism to achieve abstraction**. There can be only abstract methods in the java interface not method body. It is used to achieve abstraction and multiple inheritance in Java.
   3. **In** other words, you can say that interfaces can have methods and variables but the methods declared in interface contain only method signature, not body.
   4. **It** cannot be instantiated just like abstract class.
   5. Reasons to use Interface:
      1. **It** is used to achieve abstraction.
      2. **By** interface, we can support the functionality of multiple inheritance.
      3. **It** can be used to achieve loose coupling.
   6. **Syntax:**

**interface** <interface\_name>

{

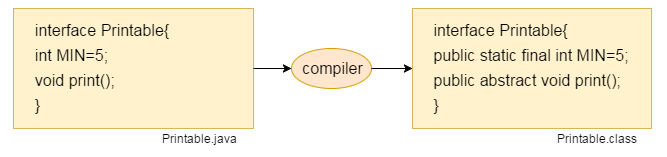
  // declare constant fields

    // declare methods that abstract

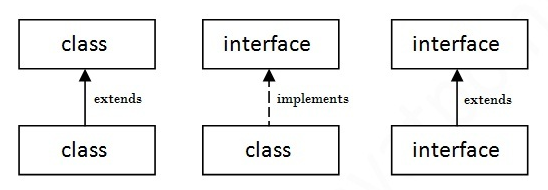
 // by default.

}

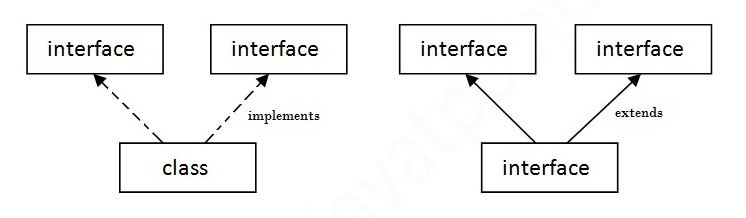
* 1. **Interface** fields are public, static and final by default, and methods are public and abstract.

****

* 1. **Understanding** relationship between classes and interfaces: As shown in the figure given below, a class extends another class, an interface extends another interface but a class implements an interface.

****

* 1. **Multiple inheritance using Interface:**
     1. If a class implements multiple interfaces, or an interface extends multiple interfaces i.e. known as multiple inheritance. Please refer to codes:



* 1. **Multiple inheritance is not supported through class in java but it is possible by interface, why?**
     + - Multiple inheritance is not supported in case of class because of ambiguity. But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class. Please refer to codes:

**interface** Printable

{

**void** print();

}

**interface** Showable

{

**void** print();

}

**class** TestInterface3 **implements** Printable, Showable

{

**public** **void** print(){System.out.println("Hello");}

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

{

TestInterface3 obj = **new** TestInterface3();

obj.print();

 }

}

* 1. **Java 8 Default Method in Interface:**
     1. Since Java 8, we can have method body in interface. But we need to make it default method.

**interface** Drawable

{

**void** draw();

**default** **void** msg(){System.out.println("default method");}

}

**class** Rectangle **implements** Drawable

{

**public** **void** draw(){System.out.println("drawing rectangle");}

}

**class** TestInterfaceDefault

{

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

{

Drawable d=**new** Rectangle();

d.draw();

d.msg();

}

}

* 1. **Java 8 Static Method in Interface**
     1. Since Java 8, we can have static method in interface.

**interface** Drawable{

**void** draw();

**static** **int** cube(**int** x){**return** x\*x\*x;}

}

**class** Rectangle **implements** Drawable{

**public** **void** draw(){System.out.println("drawing rectangle");}

}

**class** TestInterfaceStatic

{

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

Drawable d=**new** Rectangle();

d.draw();

System.out.println(Drawable.cube(3));

}

}

## What is marker or tagged interface?

* + 1. An interface which has no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

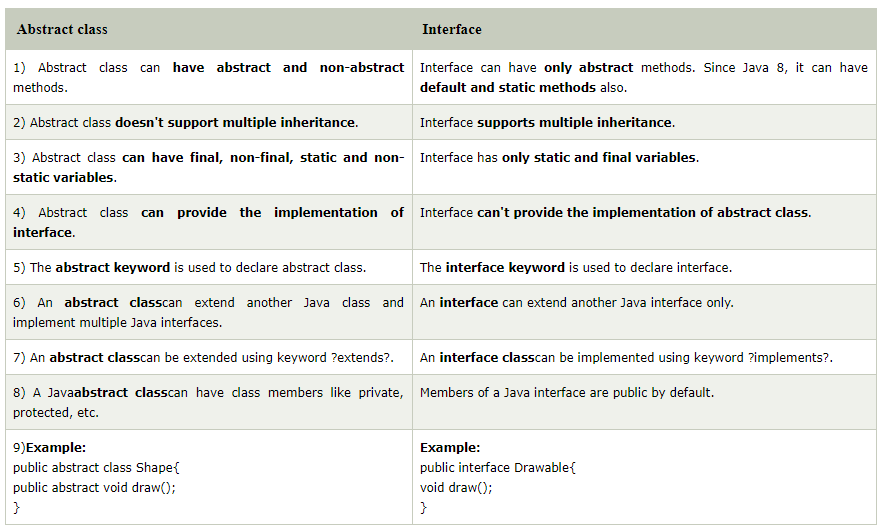
//How Serializable interface is written?

**public** **interface** Serializable

{

}

1. **Interface vs Abstract:**



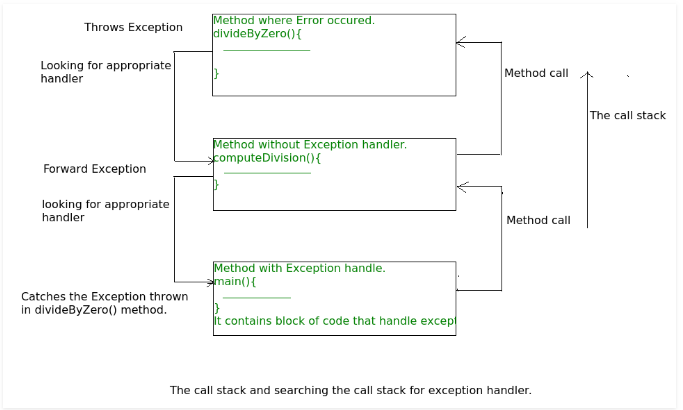
1. **Exception Handling in java:**
   1. **Exception:**
      1. An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e. at run time, that disrupts the normal flow of the program’s instructions
   2. **Error:**
      1. An Error indicates serious problem that a reasonable application should not try to catch.
   3. **Exception Hierarchy:**

All exception and errors types are sub classes of class **Throwable**, which is base class of hierarchy. One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. NullPointerException is an example of such an exception. Another branch, **Error** are used by the Java run-time system([JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/)) to indicate errors having to do with the run-time environment itself(JRE). StackOverflowError is an example of such an error.

****

* 1. **How JVM handle an Exception?**
     1. **Default Exception Handling:**Whenever inside a method, if an exception has occurred, the method creates an Object known as Exception Object and hands it off to the run-time system (JVM). The exception object contains name and description of the exception, and current state of the program where exception has occurred. Creating the Exception Object and handling it to the run-time system is called throwing an Exception. There might be the list of the methods that had been called to get to the method where exception was occurred. This ordered list of the methods is called **Call Stack**. Now the following procedure will happen.
        + **The** run-time system searches the call stack to find the method that contains block of code that can handle the occurred exception. The block of the code is called **Exception handler**.
        + The run-time system starts searching from the method in which exception occurred, proceeds through call stack in the reverse order in which methods were called.
        + If it finds appropriate handler then it passes the occurred exception to it. Appropriate handler means the type of the exception object thrown matches the type of the exception object it can handle.
        + If run-time system searches all the methods on call stack and couldn’t have found the appropriate handler then run-time system handover the Exception Object to **default exception handler,** which is part of run-time system. This handler prints the exception information in the following format and terminates program **abnormally**.

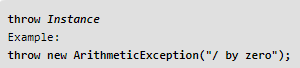
Please refer to code.



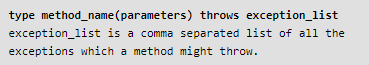
* 1. **How Programmer handles an exception?**
     1. **try- catch block**
     2. finally
     3. throw
     4. throws

**For all these exception handling techniques please see the codes.**

* 1. **throw :**
     1. The throw keyword in Java is used to explicitly throw an exception from a method or any block of code.
     2. Syntax:



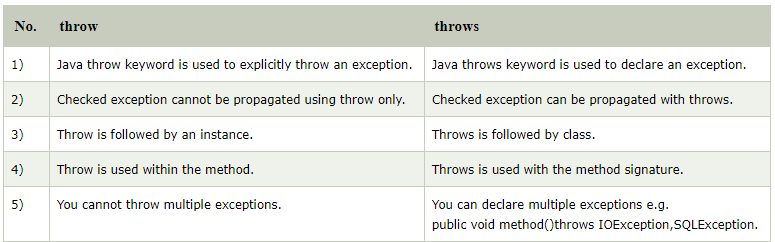
* + 1. Instance must be of type **Throwable** or a subclass of **Throwable**.
    2. The flow of execution of the program stops immediately after the throw statement is executed and the nearest enclosing **try** block is checked to see if it has a **catch** statement that matches the type of exception. If it finds a match, controlled is transferred to that statement otherwise next enclosing **try**block is checked and so on. If no matching**catch**is found then the default exception handler will halt the program.
  1. **throws:**
     1. throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed type exceptions. The caller to these methods has to handle the exception using a try-catch block.



* + 1. throws keyword is required only for checked exception and usage of throws keyword for unchecked exception is meaningless.
    2. throws keyword is required only to convince compiler and usage of throws keyword does not prevent abnormal termination of program.

**Please refer to code.**

* 1. **throw vs throws:**

****