Generic Vs Non Generic

Exception hadling in Overriding

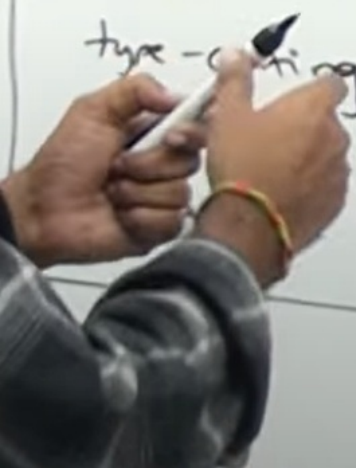
Instance flow execution In Inheritance class

Interface Vs Abstract class (After java 8)

Why Notify() , NotifyAll(), wait() defines in the Object class.

Sleep() Vs Yeild()

AL l = new AL();	AL<String> l = new AL<String>();
① It is the non Generic version of ArrayList object	① It is the Generic version of ArrayList object
② For this ArrayList, we can add any type of object and hence it is not Type-Safe.	② For this ArrayList, we can add only String type of objects and hence it is type-safe.
③ At the time of retrieval, we should perform type-casting, otherwise we will get compile-time Type-casting is the bigger in non-generic collections.	③ At the time of retrieval, it is not required to perform type-casting. Hence Type-casting head-ache are not there in Generic Collections



Exception handling in overriding.

1. **Problem 1**: If The SuperClass doesn't declare an exception
2. **Problem 2**: If The SuperClass declares an exception

Let us discuss different cases under these problems and perceived their outputs.

Problem 1: If The SuperClass doesn't declare an exception

In this problem, two cases that will arise are as follows:

- **Case 1:** If SuperClass doesn't declare any exception and subclass declare checked exception error
- **Case 2:** If SuperClass doesn't declare any exception and SubClass declare Unchecked exception no error

```
class SubClass extends SuperClass {

  // method() declaring Checked Exception IOException
  void method() throws IOException {

    // IOException is of type Checked Exception
    // so the compiler will give Error

    System.out.println("SubClass");
  }

  // Driver code
  public static void main(String args[]) {
    SuperClass s = new SubClass();
    s.method();
  }
}
```

Output:

```
mayanksolanki@Mayanks-MacBook-Air test % javac GFG.java
GFG.java:20: error: method() in SubClass cannot override method() in SuperClass
  void method() throws IOException {
       ^
  overridden method does not throw IOException
1 error
mayanksolanki@Mayanks-MacBook-Air test %
```

Case 1: If SuperClass doesn't declare any exception and subclass declare checked exception.

Example

Java

```
// Java Program to Illustrate Exception Handling
// with Method Overriding
// Where SuperClass does not declare any exception and
// subclass declare checked exception

// Importing required classes
import java.io.*;

class SuperClass {

  // SuperClass doesn't declare any exception
  void method() {
    System.out.println("SuperClass");
  }
}

// SuperClass inherited by the SubClass
class SubClass extends SuperClass {

  // method() declaring Checked Exception IOException
  void method() throws IOException {

    // IOException is of type Checked Exception
    // so the compiler will give Error

    System.out.println("SubClass");
  }
```

Case 2: If SuperClass doesn't declare any exception and SubClass declare Unchecked exception

```java
// Java Program to Illustrate Exception Handling
// with Method Overriding
// Where SuperClass doesn't declare any exception and
// SubClass declare Unchecked exception

// Importing required classes
import java.io.*;

class SuperClass {

    // SuperClass doesn't declare any exception
    void method()
    {
        System.out.println("SuperClass");
    }
}

// SuperClass inherited by the SubClass
class SubClass extends SuperClass {

    // method() declaring Unchecked Exception ArithmeticException
    void method() throws ArithmeticException
    {

        // ArithmeticException is of type Unchecked Exception
        // so the compiler won't give any error

        System.out.println("SubClass");
    }

    // Driver code
    public static void main(String args[])
    {
        SuperClass s = new SubClass();
        s.method();
    }
}
```

Output

```
SubClass
```

Now dwelling onto the next problem associated with that is if The SuperClass declares an exception. In this problem 3 cases will arise as follows:

Case 1: If SuperClass declares an exception and SubClass declares exceptions other than the child exception of the SuperClass declared Exception.

```
class SuperClass {

  // SuperClass declares an exception
  void method() throws RuntimeException {
    System.out.println("SuperClass");
  }
}

// SuperClass inherited by the SubClass
class SubClass extends SuperClass {

  // SubClass declaring an exception
  // which are not a child exception of RuntimeE
  void method() throws Exception {

    // Exception is not a child exception
    // of the RuntimeException
    // So the compiler will give an error

    System.out.println("SubClass");
  }

  // Driver code
  public static void main(String args[]) {
    SuperClass s = new SubClass();
    s.method();
  }
}
```

Output:

```
mayanksolanki@Mayanks-MacBook-Air test % javac GFG.java
GFG.java:18: error: method() in SubClass cannot override method() in SuperClass
  void method() throws Exception {
       ^
  overridden method does not throw Exception
1 error
mayanksolanki@Mayanks-MacBook-Air test %
```

Case 2: If SuperClass declares an exception and SubClass declares a child exception of the SuperClass declared Exception.

```
import java.io.*;

class SuperClass {

    // SuperClass declares an exception
    void method() throws RuntimeException
    {
        System.out.println("SuperClass");
    }
}

// SuperClass inherited by the SubClass
class SubClass extends SuperClass {

    // SubClass declaring a child exception
    // of RuntimeException
    void method() throws ArithmeticException
    {

        // ArithmeticException is a child exception
        // of the RuntimeException
        // So the compiler won't give an error
        System.out.println("SubClass");
    }

    // Driver code
    public static void main(String args[])
    {
        SuperClass s = new SubClass();
        s.method();
    }
}
```

OUTPUT:Subclasses

Case 3: If SuperClass declares an exception and SubClass declares without exception.

```
// Importing required classes
import java.io.*;

class SuperClass {

    // SuperClass declares an exception
    void method() throws IOException
    {
        System.out.println("SuperClass");
    }
}

// SuperClass inherited by the SubClass
class SubClass extends SuperClass {

    // SubClass declaring without exception
    void method()
    {
        System.out.println("SubClass");
    }

    // Driver code
    public static void main(String args[])
    {
        SuperClass s = new SubClass();
    try {
        s.method();
    } catch (IOException e) {
        e.printStackTrace();
    }
    }
```

OUTPUT:Subclasses

Conclusions:

As perceived from above 3 examples in order to handle such exceptions, the following con clusions derived are as follows:

- *If SuperClass does not declare an exception, then the SubClass can only declare unche cked exceptions, but not the checked exceptions.*
- *If SuperClass declares an exception, then the SubClass can only declare the same or ch ild exceptions of the exception declared by the SuperClass and any new Runtime Excep tions, just not any new checked exceptions at the same level or higher.*
- *If SuperClass declares an exception, then the SubClass can declare without exception.*

Instance flow example explain in hierachical

```
class parent {
    parent ()
  {   S.O.P (" parent class constructor");
  }
  {
    S.O.P(" Instance blocks of Parent");
  }
static { S.O.P ( " static block of Parent ");
       }
}
class child extends parent {
      child() {
        S.O.P(" child constructor");
      }
{
  S.O.P(" Instance blocks of child ");
}
static { S.O.P ( " static block of child ");
       }
```

```
p s v m ( string args[])
{   Parent obj = new child();
}
```

static block of Parent
Static block of child
Instance blocks of Parent
parent class constructor
Instance blocks of child
child class constructor

Static → Parent → child, static variable memory allocation

instance block Parent, Constructor

instance child, child constructor

after java 8

type	Abstract class	Interface
Constrctors	yes	No
fields		
static	yes	yes
non static	yes	no
final	yes	yes
non-final	yes	no
public	yes	yes
private	yes	no
protected	yes	no
Methods		
static	yes	yes
non static	yes	no
final	yes	no
non-final	yes	yes
public	yes	yes
private	yes	yes
protected	yes	no
defaults	no	yes

when to use Interface ?

When to use Abstact class?

Reason Why Wait, Notify, and NotifyAll are in Object Class.

1. Wait and notify is not just normal methods or synchronization utility, more than that they are **communication mechanism between two threads in Java**. And Object class is the correct place to make them available for every object if this mechanism is not available via any java keyword like synchronized.

wait() Vs sleep()

Wait()	Sleep()
Wait() method belongs to Object class.	Sleep() method belongs to Thread class.
Wait() method releases lock during Synchronization.	Sleep() method does not release the lock on object during Synchronization.
Wait() should be called only from Synchronized context.	There is no need to call sleep() from Synchronized context.
Wait() is not a static method.	Sleep() is a static method.
Wait() Has Three Overloaded	Sleep() Has Two Overloaded Methods:
public final void wait(long timeout)	public static void sleep(long millis) throws Interrupted_Execption

sleep method throw exception

2. **Locks are made available on per Object basis**, which is another reason wait and notify is declared in Object class rather then Thread class.

3. In Java in order to enter a critical section of code, Threads needs lock and they wait for lock, they don't know which threads hold lock instead they just know the lock is hold by some thread and they should wait for lock instead of knowing which thread is inside the synchronized block and asking them to release lock. this analogy fits with wait and notify being on object class rather than a thread in Java.

Difference between yield and sleep in Java

The major difference between yield and sleep in Java is that yield() method **pauses the currently executing** thread temporarily for giving a chance to the remaining waiting threads of the same priority to execute. If there is no waiting thread or all the waiting threads have a lower priority then the same thread will continue its execution.

The yielded thread when it will get the chance for execution is decided by the thread scheduler whose behavior is vendor dependent. Yield method doesn't guarantee that the current thread will pause or stop but it guarantees that CPU will be relinquished by current Thread as a result of a call to Thread.yield() method in java. See Java Concurrency in Practice for more details.