

# Core Java

---

What is wrapper class? What is their use?

- Use primitive types as objects
  - used in collection
- Data-type conversion
  - non-static methods: `intValue()`, `byteValue()`, `longValue()`, `floatValue()`, `doubleValue()`
  - static methods: `parseInt()`, `parseDouble()`, `valueOf()`, ...
- Helper methods
  - `Integer.max()`, `Integer.sum()`, ...
- Information of primitive types
  - e.g. `Integer.BYTES` = 4 bytes (int size), `Integer.MAX_VALUE` (max int), ...

Which are methods of `java.lang.Object` class? Which are native methods of object class?

- native protected `Object clone()`
  - Creates and returns a copy of this object.
- `boolean equals(Object obj)`
  - Indicates whether some other object is "equal to" this one.
- protected void `finalize()`
  - Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
- native `Class<?> getClass()`
  - Returns the runtime class of this `Object`.
- native int `hashCode()`
  - Returns a hash code value for the object.
- native void `notify()`
  - Wakes up a single thread that is waiting on this object's monitor.
- native void `notifyAll()`
  - Wakes up all threads that are waiting on this object's monitor.

- `String toString()`
  - Returns a string representation of the object.
- `void wait()`
  - Causes the current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object.
- `void wait(long timeout)`
  - Causes the current thread to wait until either another thread invokes the `notify()` method or the `notifyAll()` method for this object, or a specified amount of time has elapsed.
- `void wait(long timeout, int nanos)`
  - Causes the current thread to wait until another thread invokes the `notify()` method or the `notifyAll()` method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

What is the need of package? Which types are allowed to declare in package?

- Packages makes Java code modular. It does better organization of the code.
- Package is a container that is used to group logically related classes, interfaces, enums, and other packages.
- Package helps developer:
  - To group functionally related types together.
  - To avoid naming clashing/collision/conflict/ambiguity in source code.
  - To control the access to types.
  - To make easier to lookup classes in Java source code/docs.

Why we can not declare multiple public classes in single .java file?

- Rule: public class name must be same as file name.
- Efficient compilation process (Faster linking during import).

What is the difference between import and static import?

- `import` is used to import public types from other packages.

```
import java.util.ArrayList;  
import java.io.*;
```

- static import is used to import accessible static members of any class.

```
import static java.lang.Math.*;
import static java.lang.System.out;

class Test {
    public static void main(String[] args) {
        double result = sqrt(7.0);
        out.println(result);
    }
}
```

How can we pass argument to the method by reference? Explain with example?

- By value: Copy of argument is passed to function
  - Any changes in variable inside called function will not reflect in calling function.
  - Passing primitive types are always by value in Java.
- By reference: Reference (address) of argument is passed to function
  - Any changes in variable inside called function will reflect in calling function.
  - Passing reference types (class objects, arrays) are always by reference in Java.

```
class MyInt {
    private int a;
    // getter/setter
}

class Tester {
    // by ref
    public static void fun(MyInt x) {
```

```
        int i = x.get() * 2;
        x.set(i);
    }
    // by value
    public static void fun(int x) {
        x = x * 2;
    }
}
```

What is the difference between checked and unchecked exception?

- Java compiler expect that certain exception must be handled by the programmer -- checked exception.
  - catch block
  - throws clause
- Most of the checked exceptions arise out of the JVM. Hence Java wants programmer to be aware of them and handle them. For example:
  - File IO --> IOException
  - Database --> SQLException
  - Threads --> InterruptedException
- The other exceptions usually arise due to programmers/users mistakes and within the JVM. For example:
  - NullPointerException
  - ArithmeticException

Which are the advantages and disadvantages of generics?

- Advantages:
  - Type-safety
- Disadvantages/Limitations:
  - Doesn't work with primitive types
  - Can't create object/array of generic type
  - Can't overload based on generic type difference
  - ...

## What is difference between Comparable and Comparator?

- Comparable
  - The current object (**this**) is Comparable to the other object -- meant to be written in the class to be compared.
  - java.lang.Comparable
  - Natural ordering

```
class Student implements Comparable<Student> {  
    // ...  
    public int compareTo(Student other) {  
        int diff = this.roll - other.roll;  
        return diff;  
    }  
}
```

```
Student[] arr = { .... };  
Arrays.sort(arr); // Comparable.compareTo()
```

- Comparator
  - An object compares two other objects.
  - java.util.Comparator
  - Custom order

```
class StudentComparator implements Comparator<Student> {  
    public int compare(Student s1, Student s2) {  
        int diff = s1.marks - s2.marks;  
        return diff;  
    }  
}
```

```
Student[] arr = { .... };  
Arrays.sort(arr, new StudentComparator()); // StudentComparator.compare()  
//Arrays.sort(arr, (s1, s2) -> s1.marks - s2.marks);
```

What is difference between ArrayList and Vector? How to use ListIterator?

- V: Legacy collection 1.0
- A: Collection framework 1.2
- V: Synchronized -- Thread safe -- Slower -- Suitable in multi-thread applns
- A: Non-Synchronized -- Non thread safe -- Faster -- Suitable in single-thread applns
- V: Enumeration (later added support of Iterator)
- A: Iterator
- V: Initial size 10 & capacity grow (double)
- A: Initial size 10 & capacity grow (+half)
- V: Dynamically growing array
- A: Dynamically growing array
- ListIterator -- Bidirectional traversal

```
ListIterator<String> itr = list.listIterator();  
while(itr.hasNext()) {  
    String ele = itr.next();  
}
```

```
// ...  
}
```

```
ListIterator<String> itr = list.listIterator(list.size());  
while(itr.hasPrevious()) {  
    String ele = itr.previous();  
    // ...  
}
```

### What is fail-fast and fail-safe iterator?

- Fail Fast ... While processing iterator if any element is modified, ConcurrentModificationException is thrown.

```
java.util.ArrayList<String> list = new ArrayList<>();  
// ...  
  
// thread 1  
Iterator<String> itr = list.iterator();  
while(itr.hasNext()) {  
    String ele = itr.next();  
    // ...  
}  
  
// thread 2  
list.set(4, newValue);
```

- Fail Safe ... While processing iterator if any element is modified, modification exception is not raised.

```
List<String> list = new java.util.concurrent.CopyOnWriteArrayList<>();
// ...

// thread 1
Iterator<String> itr = list.iterator();
while(itr.hasNext()) {
    String ele = itr.next();
    // ...
}

// thread 2
list.set(4, newvalue);
```

How to make ArrayList Synchronized?

```
List<String> list = Collections.synchronizedList(new ArrayList<String>());
// internally create a proxy/wrapper that includes synchronized methods which in turn call ArrayList methods.
```

What is serialization and deserialization? What is significance of serialVersionUID?

```
```Java
class Transaction {
    static final long serialVersionUID = 2L;
    // fields ...
    // methods
}
```
```



What is relation between Thread start() and run() method?

- th.start() --> submit the JVM thread to the (jvm) scheduler.
- When scheduler schedules your thread, it (jvm) invokes run() method of the thread.

When we should create thread by implementing Runnable and extending Thread class?

- Simple answer

```
class MyThread extends Thread {  
    @Override  
    public void run() {  
        // ...  
    }  
    public void startWork() {  
        Thread t = new MyThread();  
        t.start();  
    }  
}
```

```
class MyWindow extends JFrame implements Runnable {  
    @Override  
    public void run() {  
        // ...  
    }  
    public void startWork() {  
        Thread t = new Thread(this); // this object is inherited from Runnable  
        t.start();  
    }  
}
```

- Difficult answer
  - Java was developed in Solaris (UNIX).
  - Solaris ... Two level threading model = Many to One + One to One
  - Java provide two way to create threads
    - extends Thread -- One to One (OS scheduler)
    - implements Runnable -- Many to One -- green threads (JVM scheduler)

## Deep copy vs Shallow copy

```
```Java
class Human implements Cloneable {
    int age;
    String name;
    Date birth;
    // ...
    @Override // shallow copy
    Object clone() throws ... {
        Human other = super.clone(); // Object.clone()
        return other;
    }
}
```
```

```
```Java
class Human implements Cloneable {
    int age;
    String name;
    Date birth;
    // ...
    @Override // deep copy
    Object clone() throws ... {
```

```
        Human other = super.clone(); // Object.clone()
        other.birth = this.birth.clone();
        return other;
    }
}
```

What is functional interface? Which functional interfaces are predefined and where they are used?

- Functional interface: SAM (Single Abstract Method)
  - Any number of default methods
  - Any number of static methods
- @FunctionalInterface -- compiler check for SAM - if 0 or multiple SAM, then compiler error.
- (Before Java 8) Functional Interface:
  - Comparable, Comparator, Runnable, Closeable, ...
- (In Java 8) Functional Interfaces -- java.util.function.\*
  - Predicate ... boolean test(T val);
    - used in filter() operation
  - Function ... R apply(T val);
    - used in map() operation
  - Consumer ... void accept(T val);
    - used in forEach() operation
  - Supplier ... T get();
    - used in generator
- Functional Interfaces are also used for method references.
  - Consumer cons = System.out::println;
    - cons.accept("Sunbeam"); // --> System.out.println("Sunbeam");

What is significance of filter(), map(), flatMap() and reduce() operations? In which scenarios they are used?

- java.util.Stream -- represents stream of data elements

- Immutable: stream1 --> Intermediate operation --> stream2
- Lazily evaluated: stream1.iop1().iop2().iop3().iop4().iop5().terminal();
- Stream operations
  - Intermediate operations e.g. map(), filter(), sort(), ...
  - Terminal operations e.g. forEach(), collect(), reduce(), ...

```
Integer[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9};
Stream<Integer> stream = Stream.of(array);
stream                                     // 1, 2, 3, 4, 5, 6, 7, 8, 9
    .filter(i -> i % 2 != 0)              // 1, 3, 5, 7, 9
    .map(i -> "DAC"+i)                    // DAC1, DAC3, DAC5, DAC7, DAC9
    .forEach(s -> System.out.println(s));
```

```
Integer[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9};
Stream<Integer> stream = Stream.of(array);
List<String> list = stream
    .filter(i -> i % 2 != 0)              // 1, 2, 3, 4, 5, 6, 7, 8, 9
    .map(i -> "DMC"+i)                   // 1, 3, 5, 7, 9
    .collect(Collectors.toList());       // DMC1, DMC3, DMC5, DMC7, DMC9
for(String str : list)
    System.out.println(str)
```

- <https://winterbe.com/posts/2014/03/16/java-8-tutorial/>
- <https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/>

## Functional Programming

- Concise code
- Pure functions -- Output solely depends on input (isolated -- side-effect free)
- Lazy evaluated -- better performance
- More readable
- Suitable for parallel/distributed programming

## What is use of reflection? Why reflection is slower?

- Reflection uses
  - Get metadata of the types (associated in a java.lang.Class object)
    - getName(), getSuperclass(), getMethods(), getFields(), getConstructors(), getAnnotations()
  - Invoke methods dynamically or access fields dynamically

```
Object obj = cls.newInstance();
Method m = cls.getDeclaredMethod("methodName", argTypes);
Object result = m.invoke(obj, args);
```

- Invoke method without reflection

```
ClassName obj = new ClassName();
Object result = obj.methodName(args);
```

- Reflection is slower, because it needs to find the method at runtime and then invoke it on dynamically created object.

## difference between Closeable and finalize().

- When GC collects unused object, it will invoke finalize() method.
- The finalize() should be overridden in your class to cleanup the resources.

```
class MyDao {  
    Connection con;  
    // ...  
    public void finalize() {  
        con.close();  
    }  
}
```

- However, you cannot force/guarantee the GC. Possibly if GC is delayed, your resource remains open for longer duration. Poor performance.
- Closeable implementation ensure that resource can be closed immediatly after its use.

```
class MyDao implements Closeable {  
    Connection con;  
    // ...  
    public void close() {  
        con.close();  
    }  
}
```

```
MyDao dao = new MyDao();  
// ...  
dao.close(); // resource closed immediatly
```

wait() and notify()

```
```Java  
// thread1
```

```
synchronized(obj) {  
    // ... task A  
    obj.notify();  
}  
...  
  
```Java  
// thread2  
synchronized(obj) {  
    obj.wait();  
    // ... task B (to be done after A)  
}  
...`
```