**Java Lang Package**

**Topics**

|  |  |
| --- | --- |
| **Module 14: Java Lang Package** | |
| 1. | Intro, Object class |
| 2. | String, StringBuffer, StringBuilder class |
| 3. | Wrapper classes |
| 4. | Autoboxing & AutoUnboxing |

**1. Introduction**

* For writing any java program, whether it is simple or complex, the most commonly required classes & interfaces are grouped into a separate package which is nothing but java.lang package.
* We’re not required to import java.lang package explicitly because all classes & interfaces present in lang package by default available to every java program.

**Object class** (java.lang.Object)

* The most commonly required methods for every java class (whether it is pre – defined class or customizer class) are defined in a separate class which is nothing but **Object** class.
* Every class in java is the child class of **Object** class either directly or indirectly so that **Object** class methods by default available to every java class. Hence **Object** class is considered as the root of all java classes.
* Object class defines the following 11 methods:

|  |  |
| --- | --- |
| **No.** | **Methods Name** |
| 1. | public String *toString ()* |
| 2. | public native int *hashCode ()* |
| 3. | public boolean *equals (Object obj)* |
| 4. | protected native Object *clone ()* throws CloneNotSupportedException |
| 5. | protect void *finalize ()* throws Throwable |
| 6. | public final class *getClass ()* |
| 7. | public final void *wait ()* throws InterruptedException |
| 8. | public final native void *wait (long ms)* throws InterruptedException |
| 9. | public final void *wait (long ms, int ns)* throws InterruptedException |
| 10. | public native final void *notify ()* |
| 11. | public native final void *notifyAll ()* |

**Note**: Strictly speaking, Object class contains 12 methods. The extra method is registerNatives () but this method is internally required for Object class & not available to the child classes.

**1. toString () method**

* We can use toString () method to get String representation of an Object.
* Whenever we’re trying to print Object reference, internally toString () method will be called.

e.g. Student s = new Student ();

System.out.println(s); 🡪 System.out.println(s.toString());

* If our class does not contain toString () method then Object class toString () method will be executed which is implemented as follows

**public** String toString() {

**return** getClass().getName() + "@" + Integer.*toHexString*(hashCode());

}

i.e., classname@hashcode\_in\_hexadecimal\_form e.g. Student@1888759

* We can override toString () method to provide our own String representation.

E.g. **public** String toString () {

**return** name + "----" + rollno;

}

* In all wrapper classes, in all Collection classes, String class, StringBuffer & StringBuilder classes, toString () method is overridden for meaningful string representation; hence it is highly recommended to override toString () method in our class also.

**2. hashCode () method**

* For every Object, a unique number is generated by JVM which is nothing but hashCode
* HashCode won’t represent address of Object but hashCode will be generated based on the address of the Object.
* JVM will use HashCode while saving Objects into hashing related Data structures like Hashtable, HashMap, HashSet etc.
* The main advantage of saving Objects based on hashCode is search operation will become easy (The most powerful search algorithm up to today is Hashing (O (1))).
* Based on our requirement, we can override hashCode () method in our class to generate our own hashcode.
* Overriding hashCode () method is said to be proper if for every object a unique number as hashCode is generated.
* If we’re giving the chance to Object class toString () method, it will internally call hashCode () method.
* If we’re overriding toString () method then our toString () method may not call hashCode () method.

**3. equals () method**

* We can use equals () method to check equality of two objects.

e.g. obj1.equals(obj2);

* If our class does not contain equals () method then Object class equals () method will be executed.
* Based on our requirement, we can override equals () method for content comparison. While overriding equals () method for content comparison we have to take care about the following:

1. What is the meaning of equality (i.e. whether we have to check only names or only rollno or both).
2. If we’re passing different type of Object, our equals () method should not rise **ClassCastException** i.e. we have to handle **ClassCastException** to return false.
3. If we’re passing null argument then our equals () method should not rise **NullPointerException** i.e. we have to handle **NullPointerException** to return false.