Note – Primitive datatypes can’t store null value.

**Wrapper Classes**

Q- What is wrapper class?

* Wrapper class is a class that convert(or wrap) primitive type value into wrapper class object.

Q- Why we need wrapper class?

* Because we can’t apply OOPs on primitive type values, if we want to apply OOPs on primitive type value then we have to convert them into wrapper class object.
* For each primitive datatype we have its respective wrapper class.
* Present in java.lang package.

|  |  |
| --- | --- |
| Primitive data types | Wrapper Class |
| byte | Byte |
| short | Short |
| long | Long |
| int | Integer |
| float | Float |
| double | Double |
| char | Character |

Features of Wrapper class

- they are final, immutable and its object is thread-safe.

- In all wrapper class toString(), hashcode() and equals() all are overridden, it means the value inside that address will do all the work.

- All wrapper class implements comparable interface, this is why wrapper class objects are mutually comparable.

Boxing operation- converting primitive value into their respective wrapper class object.

For ex - Integer i1 = new Integer(10) ;

Unboxing operation- converting wrapper class object into their respective primitive type value.

For ex- int a = i1.intValue() ;

From JDK 1.5(Java 5) java supports auto-boxing and auto-unboxing.

Auto-boxing

Integer i1 = 10 ;

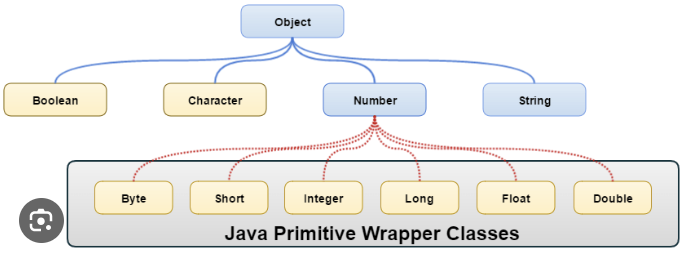
Auto-unboxing

int a = i1 ;

* This auto boxing and auto unboxing is done by the JVM.

Note – you need to know only auto-boxing and autoUnboxing.

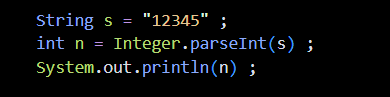
**Wrapper Class hierarchy**

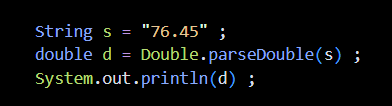
****

Number class is Abstract class, except Number class, Rest all are concrete and final class and tostring(), hashCode(), equals() is overridden in all classes.

**String Parsing**

Extracting numbers from String format is known as String parsing.



****

**Object class**

* It is a supermost class in java.
* All classes directly or indirectly inherits from Object class.
* There are total 11 concrete method present in Object class.
* And it is present in java.lang package

11 Methods of Object Class

toString()

public String toString(){

return fullyQualifiedClassName@hexaDecimalAddress ;

}

* Its return type is String.
* It will return FullyQualifiedClassName@HexadecimalAddress of an object.

hashCode()

* Return type is int.
* hashCode() will return a int value which is made by converting hexa-decimal address of the object by an algorithm.This int value is known as hashCode value of an object.
* JVM internally uses this hashcode only to uniquely identitfy the object.Every Object’s hashCode will be different, because hexa-decimal address of an object will be different.

Note – hashCode does not mean address always, it is just an int value which can be made by converting anything like address(here) and key(in hashMap).

equals()

public boolean equals(Object args){

return this.hashCode() == args.hashCode() ;

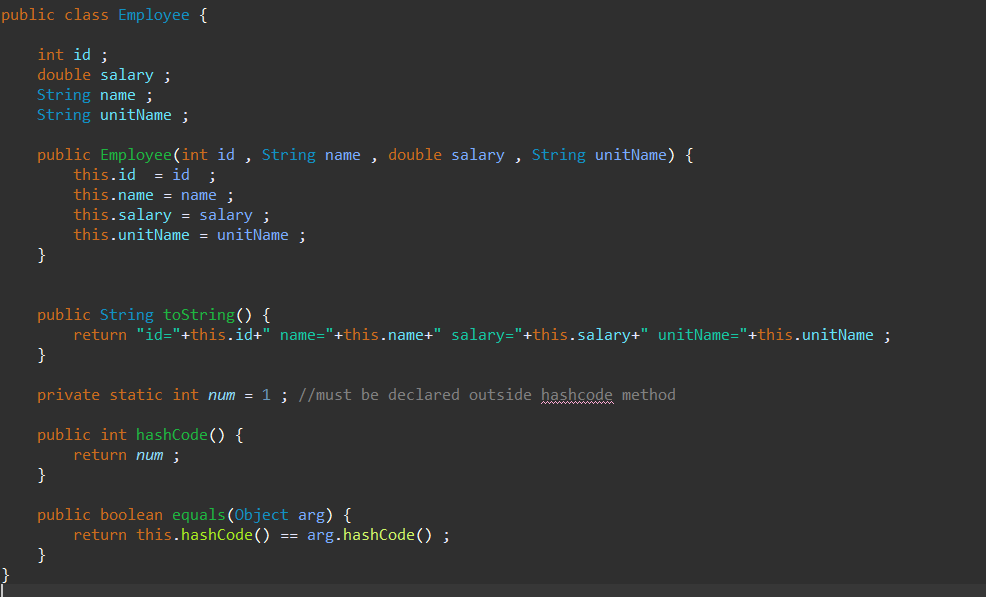
}

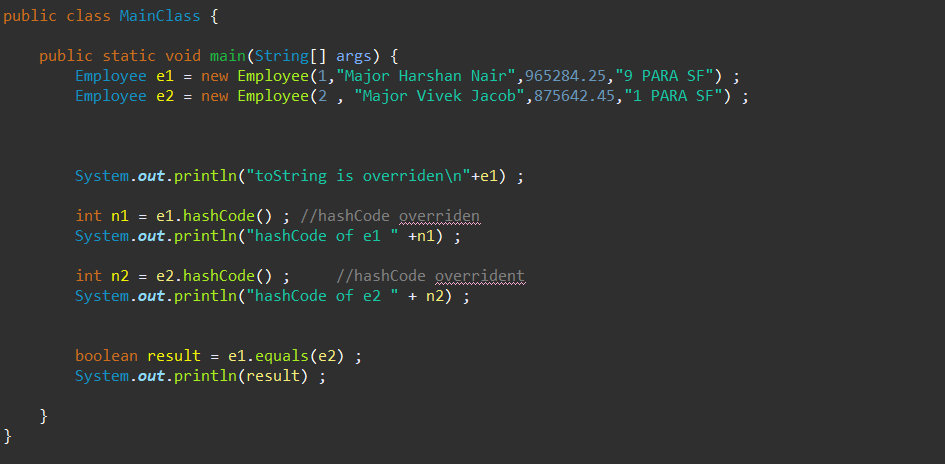
* Return type is boolean.
* equals() method internally calls hashCode() method on both current object and passed object and compare their hashcode value, if same return true(which means both object are same) otherwise false(both objects are different).

Note- We have to override both equals() method and hashCode() together. Because if we override only equals() method it will call original hashCode() method which return hashCode Value only. And if we override only hashCode() method then equals() method will call original hashCode() method not the overridden one that why override both the method together.

We can override all the three methods toString(), hashCode() and equals.

**Example program** of Overriden 🡪 toString() , hashCode() , equals()





**String**  (study from notebook)

1 and 2 page

Immutable

* In Java, when we say a String is immutable, it means the sequence of character cannot be modified or changed, once a String object is created,
* Operations like concatenation in current string or re-initialization of current string will create a new String object which will be stored in new memory address rather than the changes happening in old memory address.

This leads to memory wastage.

In String class toString(), equals(), hashCode() are overridden.

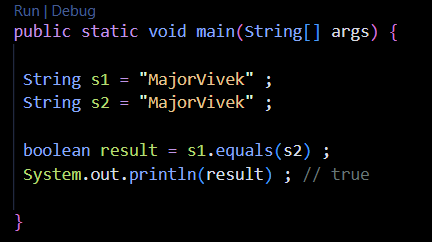
String class implements comparable interface that why its object is mutually comparable.

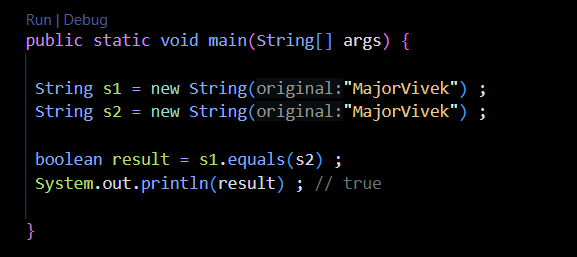
**Predefined Methods in String Class**

1. equals() method

originally equals() method orginally compare two object address but in case of String it is overridden, so here values, which are present inside the address will be compared.

Syntax – str1.equals(str2)





1. length()

* used to know the size of String.
* returns type is int.

Syntax – s.length()

1. charAt()

* used to return the character at particular index.
* Return type is char.

Syntax- s.charAt(i)

In case of non-primitive datatypes == operator compares address because reference variable holds the address inside them, so == operator compares address.

In case of primitive datatypes == operator, compares value because primitive variable holds value inside them, so == operator compare value.

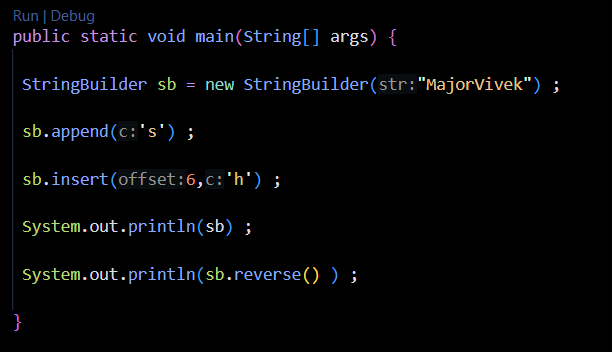
**StringBuffer and StringBuilder**

In java, sequence of character can be stored in three ways using String, StringBuilder, StringBuffer.

What is StringBuilder and StringBuffer

* StringBuffer and StringBuilder are predefined classes used to store sequence of character and is present in java.lang package.
* Both are mutable , final, and toString() method is overridden but not hashCode() and equals().

Syntax and program look in notebook



Output - MajorVhiveks

skevihVrojaM

Q- Why StringBuffer and StringBuilder is needed if String was already available ?

String was immutable, so problem was any modification in character sequence of String will lead to new object creation which will be stored in new memory address, which leads to memory wastage.

This problem was solved in StringBuilder and StringBuffer.

StringBuffer and StringBuilder are mutable, means they allow modifications in character sequence and changes will happen in same memory address without creating new object.

| **StringBuffer** | **StringBuilder** |
| --- | --- |
| StringBuffer is present since JAVA 1.0 | StringBuilder was introduced in Java 5 (JDK 1.5). |
| StringBuffer is synchronized. This means that multiple threads cannot call the methods of StringBuffer simultaneously. | StringBuilder is asynchronized. This means that multiple threads can call the methods of StringBuilder simultaneously. |
| Due to synchronization, StringBuffer is called a thread safe class. | Due to its asynchronous nature, StringBuilder is not a thread safe class. |
| Due to synchronization, StringBuffer is lot slower than StringBuilder. | Since there is no preliminary check for multiple threads, StringBuilder is a lot faster than StringBuffer. |