java.lang

Shristi Technology Labs

Contents

- Object
- Wrapper classes
- Conversions
- String
- StringBuffer
- String Builder

Few classes in Lang Package

- Object
- Math
- String
- StringBuffer
- StringBuilder
- Throwable
- Exception
- Thread
- All Wrapper classes(eg.Integer,Float)
- Runtime

Few Interfaces in Lang Package

- Runnable
- Comparable
- Cloneable

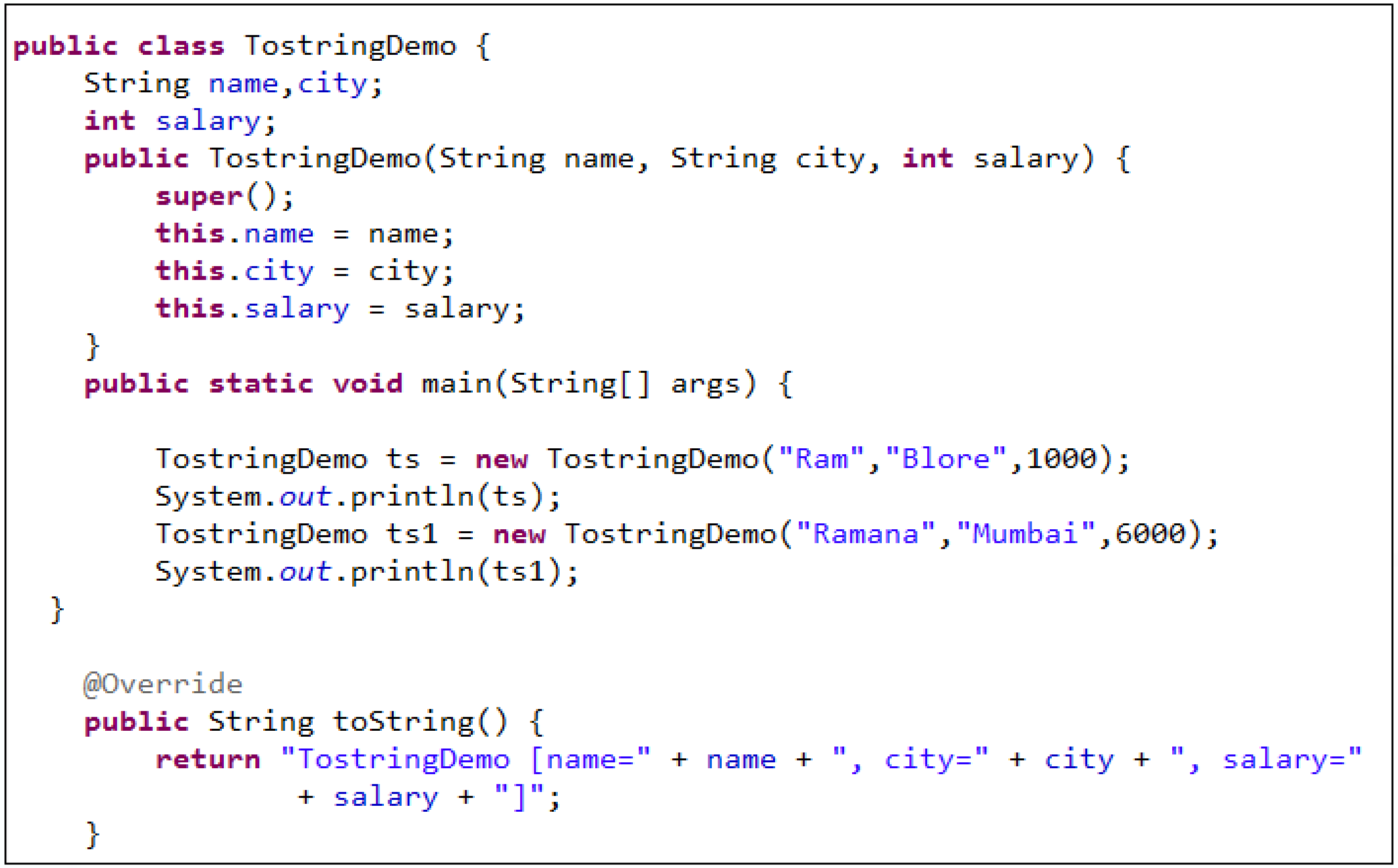
Object

- Is the top level class.
- All the classes extend this.
- Has few methods
  - hashcode() and equals() → in util for Set
  - clone()
  - getClass() → with reflection
  - wait(),notify() and notifyAll()
  - finalize() → garbage collection
  - toString()

Use of toString()

- While printing an object will get hashcode value.
- Override toString() method to get a proper output
- Used by Exception classes

Example for toString()



```
public class TostringDemo {
    String name,city;
    int salary;
    public TostringDemo(String name, String city, int salary) {
        super();
        this.name = name;
        this.city = city;
        this.salary = salary;
    }
    public static void main(String[] args) {

        TostringDemo ts = new TostringDemo("Ram","Blore",1000);
        System.out.println(ts);
        TostringDemo ts1 = new TostringDemo("Ramana","Mumbai",6000);
        System.out.println(ts1);
  }

    @Override
    public String toString() {
        return "TostringDemo [name=" + name + ", city=" + city + ", salary="
                + salary + "]";
    }
```

Cloning

- Is creating a copy of the original object.
- Is field by field copy
- Cloned object has separate memory address
- Types of cloning supported by Java
  - **Shallow Cloning**
  - **Deep Cloning**

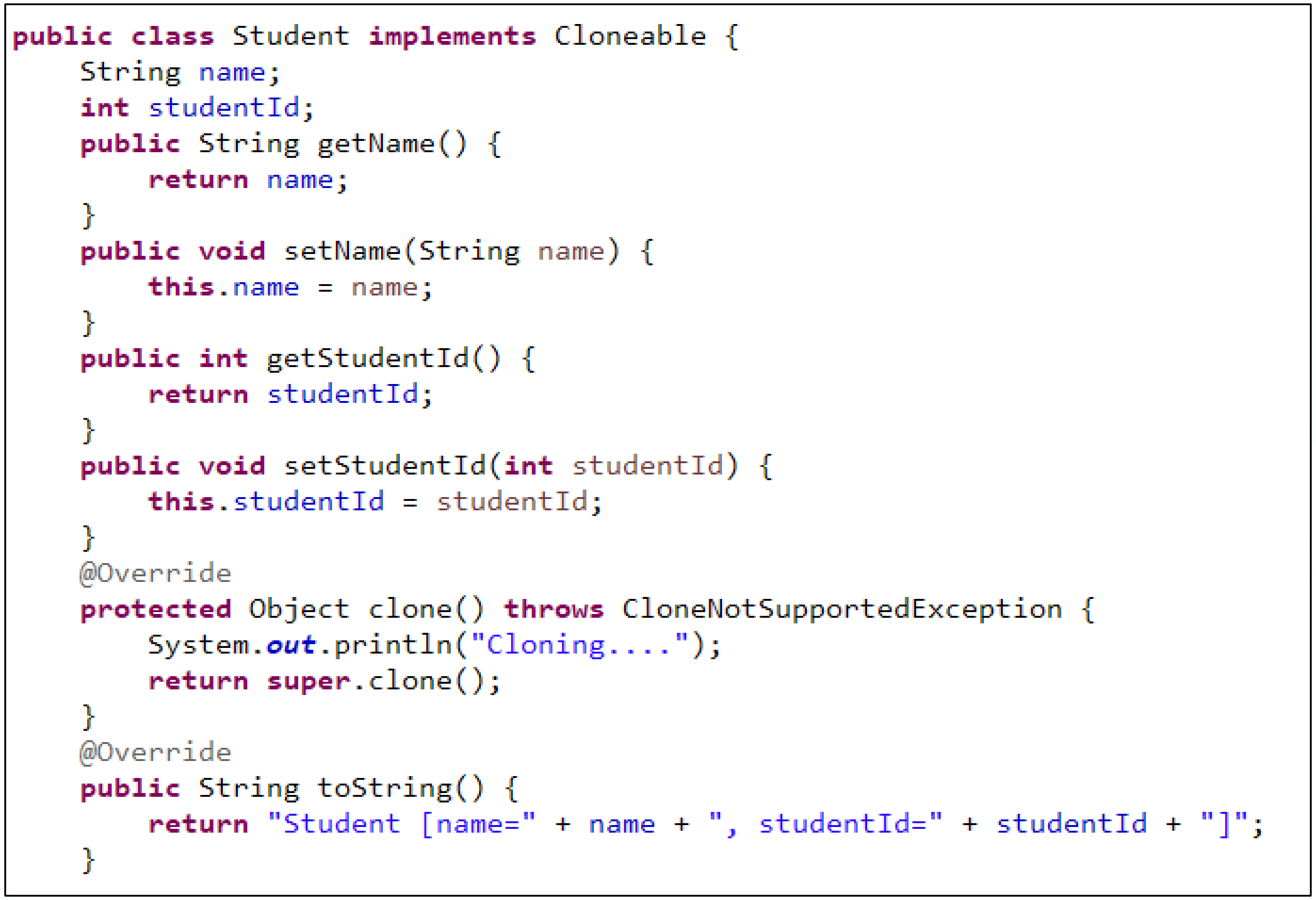
Steps in cloning

- Class must implement **Cloneable**
- Class must override **clone** method
- No constructor is called on the object being cloned.

Shallow Cloning

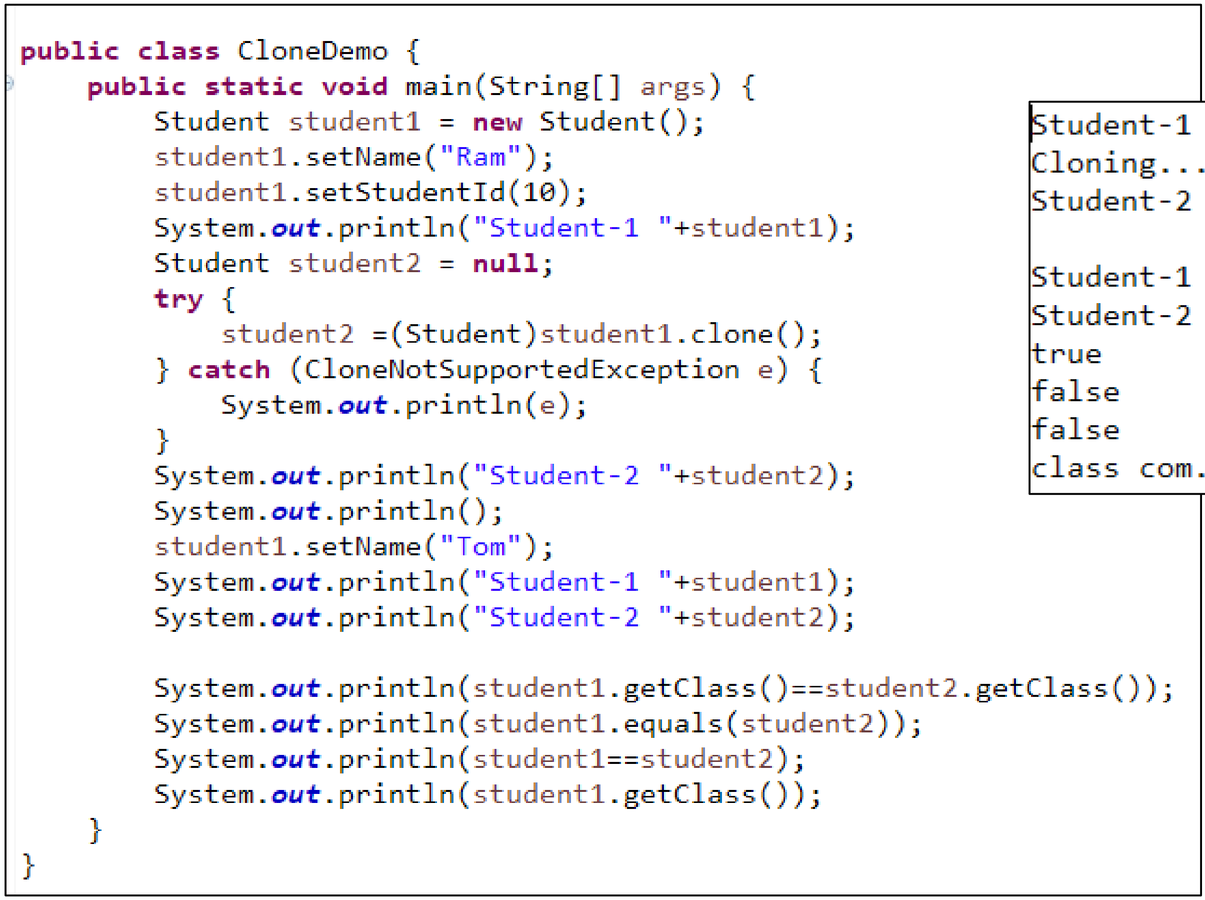
- **default implementation in java**.
- If you are not cloning all the object data types (not primitives), then it is a shallow copy
- If the class has **only primitive data type members**,
  - then a completely new copy of the object will be created
  - the reference to the **new object copy will be returned**.

Student.java



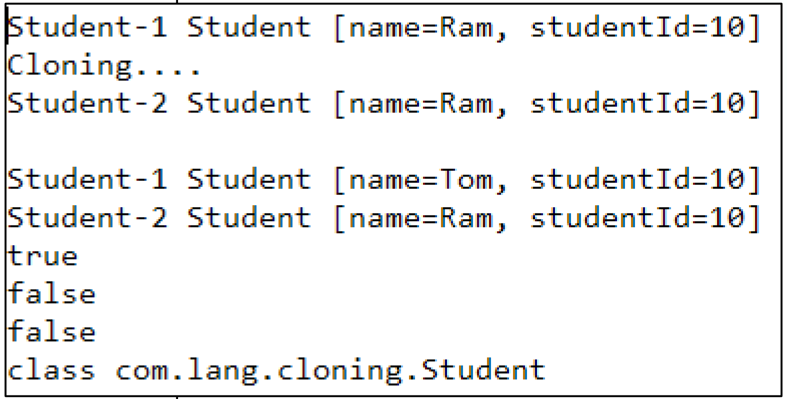
```java
public class Student implements Cloneable {
    String name;
    int studentId;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getStudentId() {
        return studentId;
    }
    public void setStudentId(int studentId) {
        this.studentId = studentId;
    }
    @Override
    protected Object clone() throws CloneNotSupportedException {
        System.out.println("Cloning....");
        return super.clone();
    }
    @Override
    public String toString() {
        return "Student [name=" + name + ", studentId=" + studentId + "]";
    }
```

CloneDemo.java



```java
public class CloneDemo {
    public static void main(String[] args) {
        Student student1 = new Student();
        student1.setName("Ram");
        student1.setStudentId(10);
        System.out.println("Student-1 "+student1);
        Student student2 = null;
        try {
            student2 =(Student)student1.clone();
        } catch (CloneNotSupportedException e) {
            System.out.println(e);
        }
        System.out.println("Student-2 "+student2);
        System.out.println();
        student1.setName("Tom");
        System.out.println("Student-1 "+student1);
        System.out.println("Student-2 "+student2);

        System.out.println(student1.getClass()==student2.getClass());
        System.out.println(student1.equals(student2));
        System.out.println(student1==student2);
        System.out.println(student1.getClass());
    }
}
```

Output



```
Student-1 Student [name=Ram, studentId=10]
Cloning....
Student-2 Student [name=Ram, studentId=10]

Student-1 Student [name=Tom, studentId=10]
Student-2 Student [name=Ram, studentId=10]
true
false
false
class com.lang.cloning.Student
```

Hashcode and equals

- Every object has access to the *equals()* method because it is inherited from the *Object* class.

- Default implementation - compares the memory addresses of the objects.

- A class can override the *equals()* method from *Object class*.
- In this case, the class should also override *hashCode()*.
- Else a violation of the general contract for Object.hashCode will occur

Contract

- ***If two objects are equal according to the equals(Object) method, then calling the hashCode() method on each of the two objects must produce the same integer result.***
- ***It is NOT required that if two objects are unequal according to the equals(Object) method, then calling the hashCode() method on each of the two objects must produce distinct integer results.***

Condition	Required	Not required (but allowed)
x.equals(y) == true	x.hashCode() ==y.hashCode()	
x.hashCode() == y.hashCode()		x.equals(y) == true
x.equals(y) == false		No Hashcode requirements

Wrapper Classes

- Wrapper classes are inbuilt classes which convert the primitive data types into objects.
- All the methods of the wrapper classes are static.

Wrapper classes

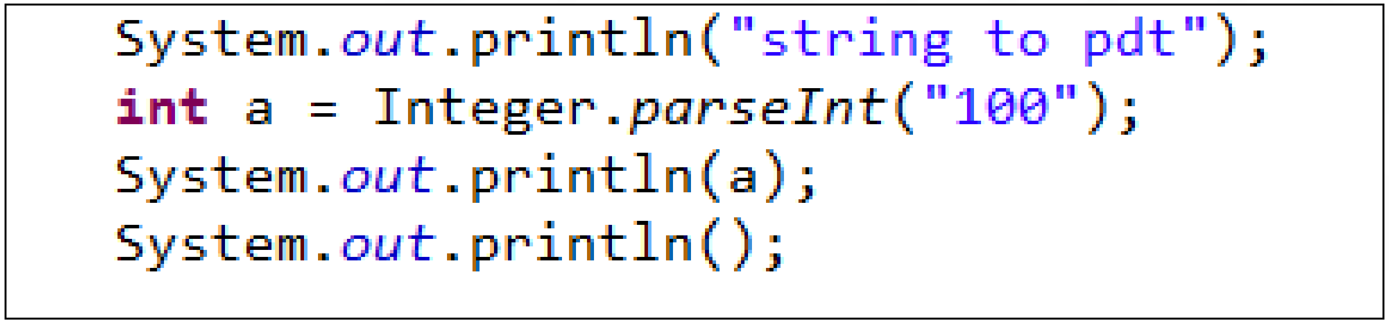
- Boolean
- Byte
- Short
- Integer
- Long
- Double
- Float

Conversions

- String to Primitive Data Type
- PDT to String
- PDT to Object
- Object to PDT
- String to Object
- Object to String

String to PDT

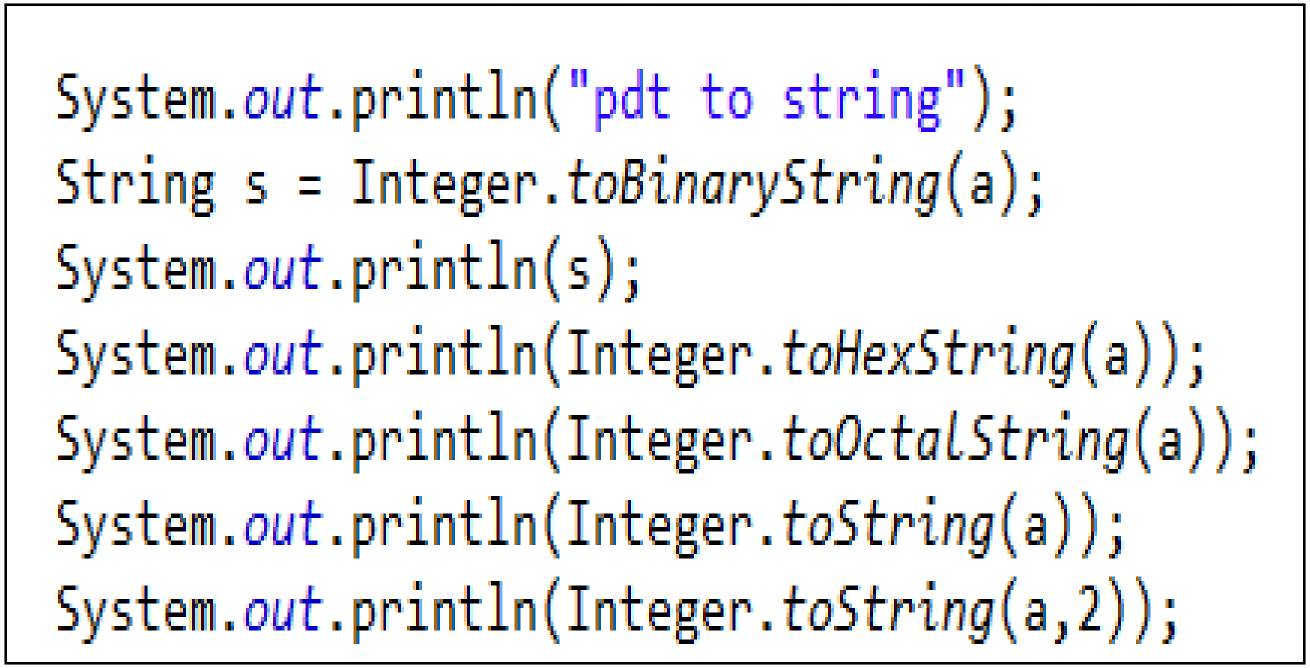
xxx value = XXX.parseXXX(String a);



```
System.out.println("string to pdt");
int a = Integer.parseInt("100");
System.out.println(a);
System.out.println();
```

PDT to String

Has static Methods only.
String s = XXX.toString(pdt);


```
System.out.println("pdt to string");
String s = Integer.toBinaryString(a);
System.out.println(s);
System.out.println(Integer.toHexString(a));
System.out.println(Integer.toOctalString(a));
System.out.println(Integer.toString(a));
System.out.println(Integer.toString(a,2));
```

PDT to Object- Autoboxing

- Used to wrap a primitive data to Object

Using AutoBoxing

```
int x =10;
Integer y = x ;
System.out.println (“y = ” + y ) ;  // Integer Object
```

Object to PDT - AutoUnboxing

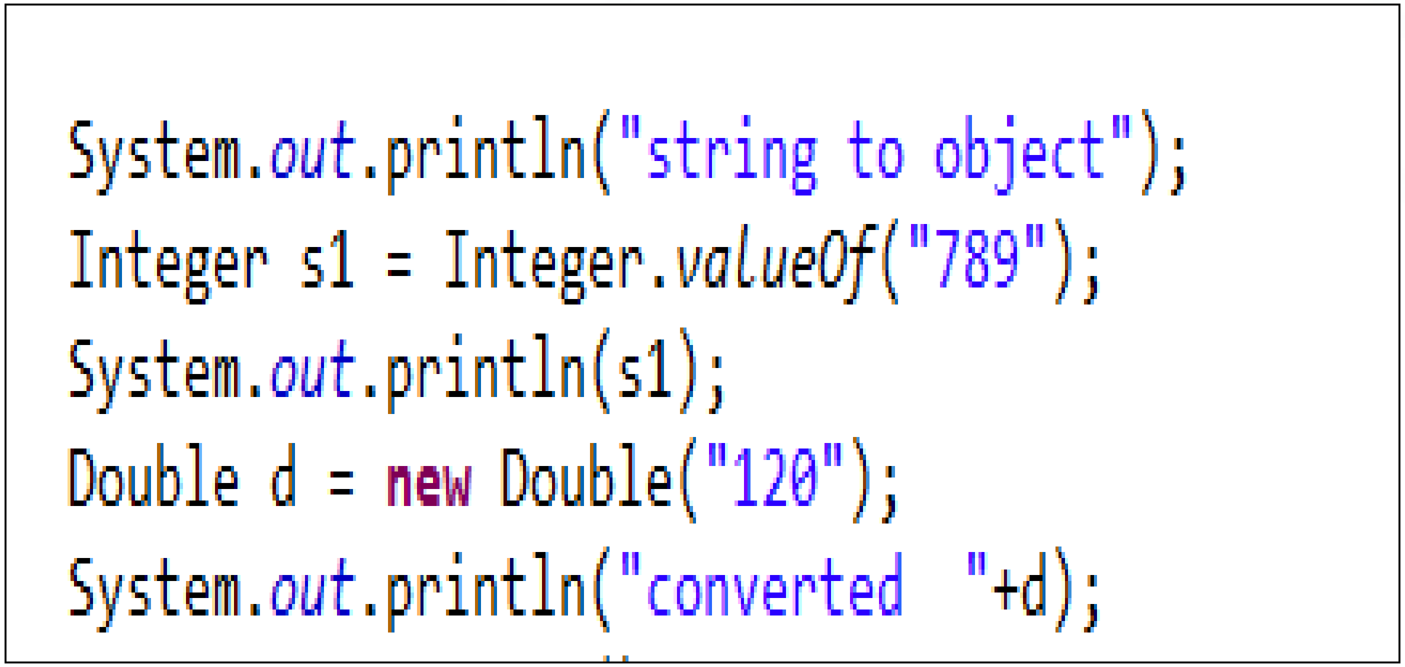
- Used to unwrap an Object to primitive data type

Using Auto UnBoxing

```
Integer y = new Integer(100) ;
 int x =y;
 System.out.println (“x = ” + x ) ;  // PDT
```

String to Object

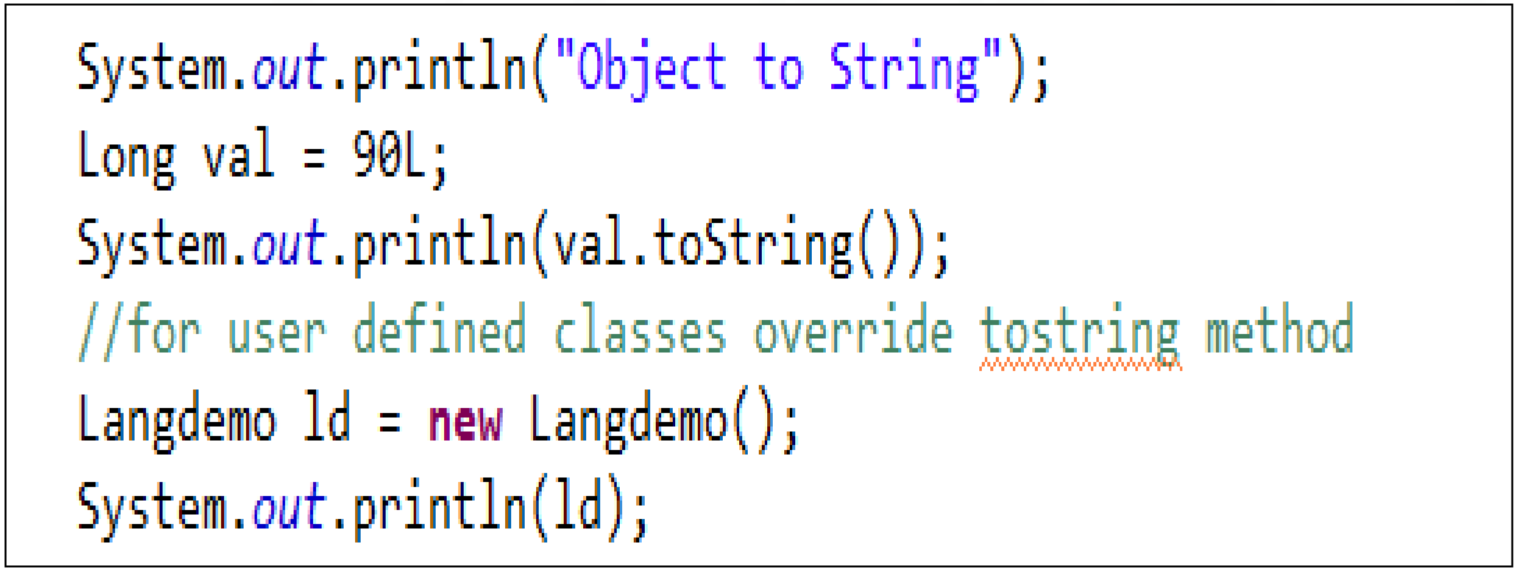
XXX value = XXX.valueOf(String a);



```
System.out.println("string to object");
Integer s1 = Integer.valueOf("789");
System.out.println(s1);
Double d = new Double("120");
System.out.println("converted  "+d);
```

Object to String

String value = instancevariable.toString();



```
System.out.println("Object to String");
Long val = 90L;
System.out.println(val.toString());
//for user defined classes override tostring method
Langdemo ld = new Langdemo();
System.out.println(ld);
```

String

- This class represents character strings.
- An immutable sequence of characters.
- Can’t use insert, append, delete methods in a String object

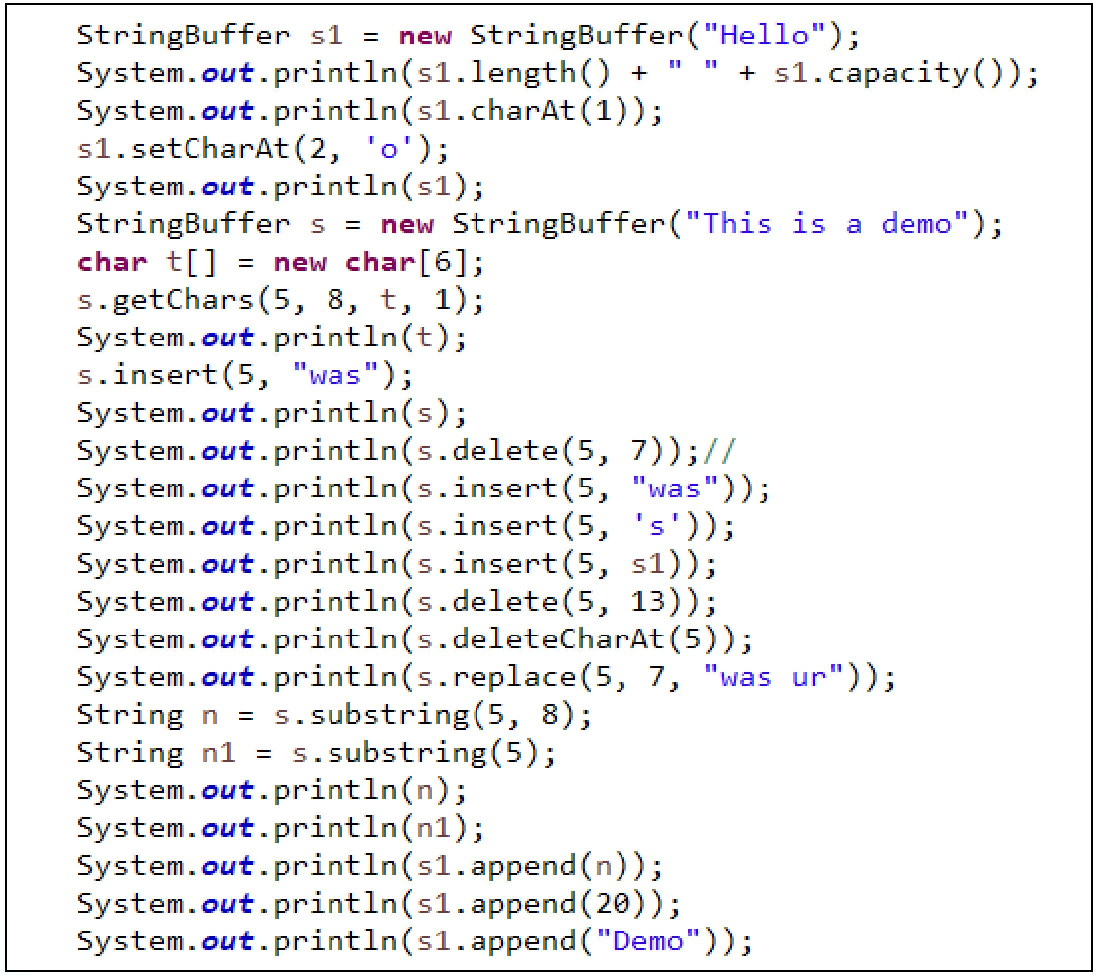
Example - String

```
String s = "hello!How are you?";
System.out.println("Char at  6 " + s.charAt(6));
System.out.println("Ends with u " + s.endsWith("u"));

char[] c = new char[9];
s.getChars(5, 13, c, 1);
for (char v : c)
    System.out.print(v);

System.out.println("Index of o " + s.indexOf('o'));
System.out.println("Index of o " + s.indexOf('o', 5));
System.out.println("Index of o " + s.lastIndexOf('o'));
String sub = s.substring(5, 12);
System.out.println("substring " + sub);
String s1 = "Hello";
String s2 = "Hello";
System.out.println(" string literals :" + s1.equals(s2));
System.out.println(" String ref " + (s1 == s2));
String s3 = new String("Hello");
String s5 = new String("Hello");
System.out.println("string obj :" + (s3.equals(s5)));
System.out.println(" string ref :" + (s5 == s3));
System.out.println(" string ref :" + (s1 == s3));
```

StringBuffer

- A mutable sequence of characters.
- StringBuffer methods are synchronized, so it is thread safe
- Can be used with multiple threads.

Example - String Buffer

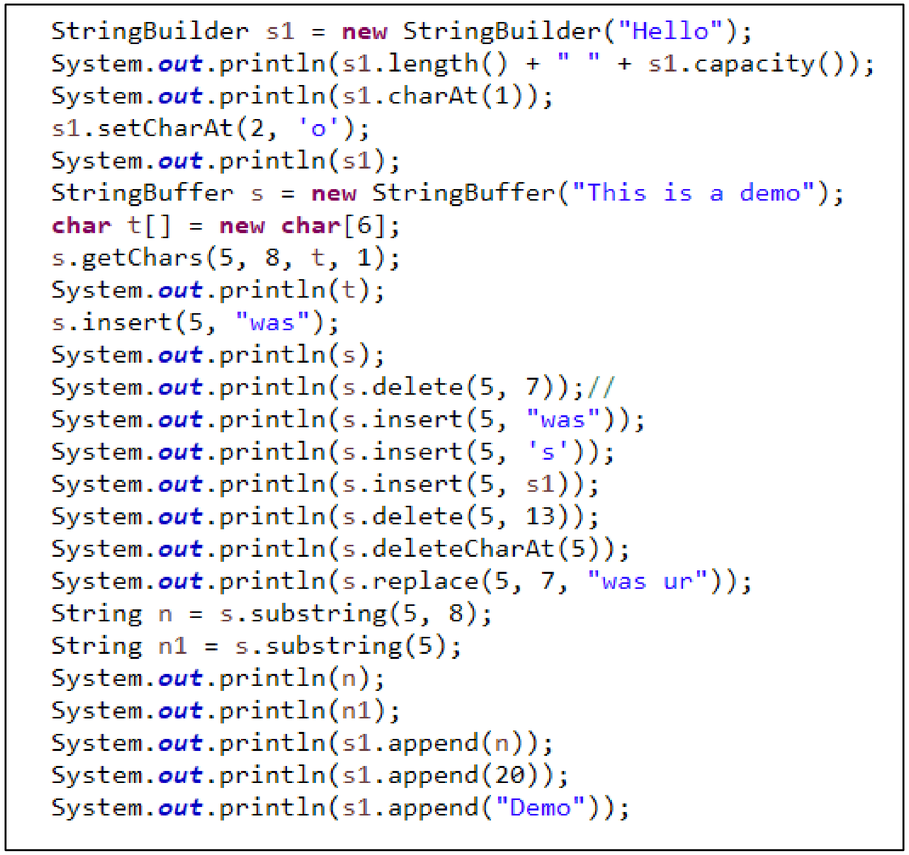


```java
StringBuffer s1 = new StringBuffer("Hello");
System.out.println(s1.length() + " " + s1.capacity());
System.out.println(s1.charAt(1));
s1.setCharAt(2, 'o');
System.out.println(s1);
StringBuffer s = new StringBuffer("This is a demo");
char t[] = new char[6];
s.getChars(5, 8, t, 1);
System.out.println(t);
s.insert(5, "was");
System.out.println(s);
System.out.println(s.delete(5, 7));//
System.out.println(s.insert(5, "was"));
System.out.println(s.insert(5, 's'));
System.out.println(s.insert(5, s1));
System.out.println(s.delete(5, 13));
System.out.println(s.deleteCharAt(5));
System.out.println(s.replace(5, 7, "was ur"));
String n = s.substring(5, 8);
String n1 = s.substring(5);
System.out.println(n);
System.out.println(n1);
System.out.println(s1.append(n));
System.out.println(s1.append(20));
System.out.println(s1.append("Demo"));
```

StringBuilder

- A mutable sequence of characters.
- An API similar to StringBuffer.
- StringBuilder methods are not synchronized, so not thread safe
- Cannot be used with multiple threads.
- is faster

Example - StringBuilder



```java
StringBuilder s1 = new StringBuilder("Hello");
System.out.println(s1.length() + " " + s1.capacity());
System.out.println(s1.charAt(1));
s1.setCharAt(2, 'o');
System.out.println(s1);
StringBuffer s = new StringBuffer("This is a demo");
char t[] = new char[6];
s.getChars(5, 8, t, 1);
System.out.println(t);
s.insert(5, "was");
System.out.println(s);
System.out.println(s.delete(5, 7));//
System.out.println(s.insert(5, "was"));
System.out.println(s.insert(5, 's'));
System.out.println(s.insert(5, s1));
System.out.println(s.delete(5, 13));
System.out.println(s.deleteCharAt(5));
System.out.println(s.replace(5, 7, "was ur"));
String n = s.substring(5, 8);
String n1 = s.substring(5);
System.out.println(n);
System.out.println(n1);
System.out.println(s1.append(n));
System.out.println(s1.append(20));
System.out.println(s1.append("Demo"));
```

Thank You