EXPLICIT AND AUTOMATIC STRING CONVERSION IN JAVA

Data with other data types can be converted to String using explicit and automatic ways in Java. Explicit String conversion ways include:

- Use of the toString() method of an object, especially wrapper classes.
- Use of the static overloaded method String.valueOf, passing in the other data type value.
- Use of a String constructor to convert a byte array as a String.

Automatic String conversion of other data type values happen when objects are used in the context of a String like:

- An object or primitive passed as an argument to a println().
- An object or primitive written as the second operand of the “+ operator” when the first operand is a String.

Code for examples

We will use the below variable declarations for our examples inside a class StringCheck:

```
public class StringCheck {

  public static void main(String[] args) {

    int i=10;

    char[] myCharArray = { 'a', 'b', 'c' };

    int[] myIntArray={1,2,3,4};

    byte myByteArray[] = "abcd".getBytes();

//Code for each case will go here.

  }

}
```

Use of the toString() method

The toString() method is part of the Object class that should return a String representation of the object and hence inherited by all classes in Java. It is recommended that all subclasses override this method. Invoking toString() on a null reference will cause NullPointerException to be thrown.

Example:

```
System.out.println(new Integer(i).toString());

System.out.println(new StringCheck().toString());

System.out.println(myCharArray.toString());

System.out.println(myIntArray.toString());

System.out.println(myByteArray.toString());
```

This will print:

10

StringCheck@1a8c4e7

[C@9664a1

[I@13e8d89

[B@1be2d65

The toString() method is part of the Object class and hence inherited by all classes in Java, but not primitives. The wrapper classes provide a mechanism to "wrap" primitive values in an object so that the primitives can be included in activities only for objects. Hence we need to wrap our primitive inside the corresponding wrapper class and then call the toString() method. Wrapper classes in Java (Integer, Long, Float etc) have overridden this method to return the String representation of the primitive it holds. Read more about wrapper classes athttp://javajee.com/wrapper-classes-auto-boxing-and-auto-unboxing.

Since we have not overridden toString for our class StringCheck, it uses the inherited default implementation in the Object class. The toString method for class Object returns a string consisting of the name of the class of which the object is an instance, the at-sign character `@', and the unsigned hexadecimal representation of the hash code of the object.

In case of Arrays, toString prints a symbol [followed by an alphabet like C, B etc. denoting the type, followed by the at-sign character '@', and the unsigned hexadecimal representation of the hash code of the object. It prints something like [C@9664a1 for char[] and [B@addbf1 for byte[]. Now let us see in detail what [C@9664a1 mean for an array.

- The [means array.
- The C means char.
- The @ separates the type from the ID.
- The hex digits are an object ID or hashcode.
  Below is a list of possible values for most types.

- B - byte
- C - char
- D - double
- F - float
- I - int
- J - long
- S - short
- Z - boolean
- [- one [for every dimension of the array

Use of String.valueOf

The static overloaded method String.valueOf returns the string representation of the argument data.

String.valueOf() is overloaded for all the primitive types, char array and for the type Object.

For the primitive types, valueOf() returns the string representation of the of the value it contains.

For objects including arrays, but excluding char[], if the argument is null, then a string equal to "null"; otherwise, the value of obj.toString() is returned.

For char array, if the argument is null, then a NullPointerException is thrown; otherwise it will print the contents of the char array.

Example:

```
System.out.println(String.valueOf(i));

System.out.println(String.valueOf(new StringCheck()));

System.out.println(String.valueOf(myCharArray));

System.out.println(String.valueOf(myIntArray));

System.out.println(String.valueOf(myByteArray));
```

This will produce the output:

10

StringCheck@13e8d89

abc

[I@1be2d65

[B@9664a1

You can see that except for character arrays, everything is printed as is for toString(); for char arrays, the contents are printed instead.

Converting a byte array to String using String constructor

Using the String.valueOf or the toString method of object will work for primitives and wrappers, but not for objects and arrays. For most objects, we need to override the toString. We also saw that a char array contents are converted with String.valueOf.

We can convert a byte array contents to String using a String constructor as:

```
System.out.println(new String(myByteArray).toString());
```

This will print:

abcd

Note that toString is overridden for the String class.

You can also specify the charset for conversion using another overloaded String constructor:

```
System.out.println(new String(myByteArray, "UTF-8").toString());
```

This can throw an UnsupportedEncodingException. To know more about charsets and endocings, refer to javajee.com/charset-and-encodings.

Automatic string conversion inside println method

Java converts any object into its string representation inside a println() by calling one of the overloaded versions of the String.valueOf() method.

Example:

```
System.out.println(i);

System.out.println(new StringCheck());
```

```
System.out.println(myCharArray);

System.out.println(myIntArray);

System.out.println(myByteArray);
```

This will output:

```
10

StringCheck@1a8c4e7

abc

[I@1be2d65

[B@9664a1
```

Note that the output is exactly same as that of String.valueOf.

Automatic String conversion when using String concatenation operator +

Java doesn't allow operator overloading yet, + is overloaded for class String. When you add a non-string operand such as an integer or char to a String, the non-string operand is converted to a string and string concatenation happens. String conversion using concatenation operator for object references, which include all array types, is defined as follows: If the reference is null, it is converted to the string "null". Otherwise, the conversion is performed as if by an invocation of the toString method of the referenced object with no arguments; but if the result of invoking the toString method is null, then the string "null" is used instead. The string concatenation operator + makes use of toString directly whereas println makes use of String.valueOf and gives special treatment to char arrays. StringBuffer.append() also gives special treatement for char[] through overloading.

Example:

```
System.out.println(""+i);

System.out.println(""+new StringCheck());

System.out.println(""+myCharArray);

System.out.println(""+myIntArray);

System.out.println(""+myByteArray);
```

This will print:

10

StringCheck@1729854

[C@cf2c80

[I@1be2d65

[B@9664a1

Note that the output is exactly same as that of toString() invocation.

Source : http://javajee.com/explicit-and-automatic-string-conversion-in-java