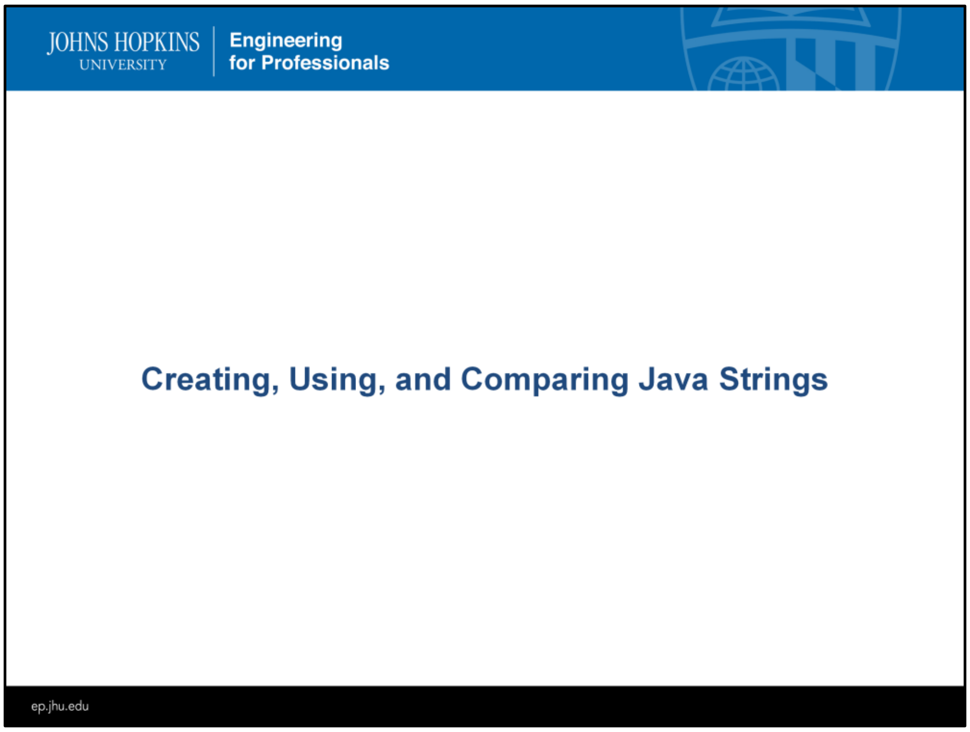


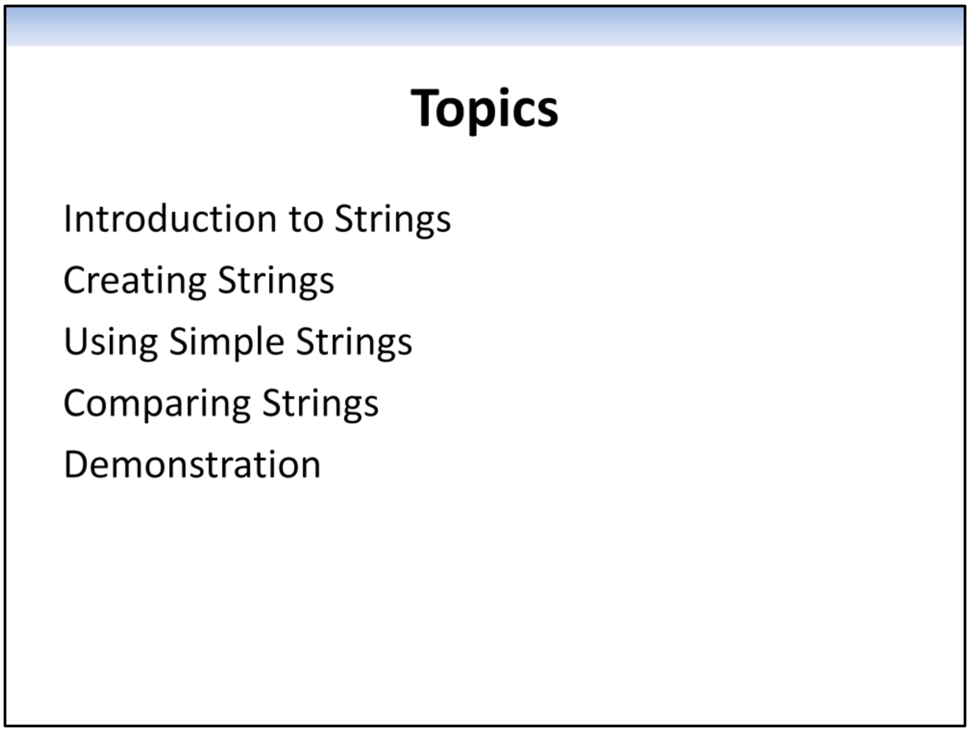
JOHNS HOPKINS UNIVERSITY | Engineering for Professionals

Creating, Using, and Comparing Java Strings

ep.jhu.edu

This presentation is a brief discussion on creating, using, and comparing Java strings.

Strings are very commonly used objects in Java so lets get to it.



Topics

Introduction to Strings

Creating Strings

Using Simple Strings

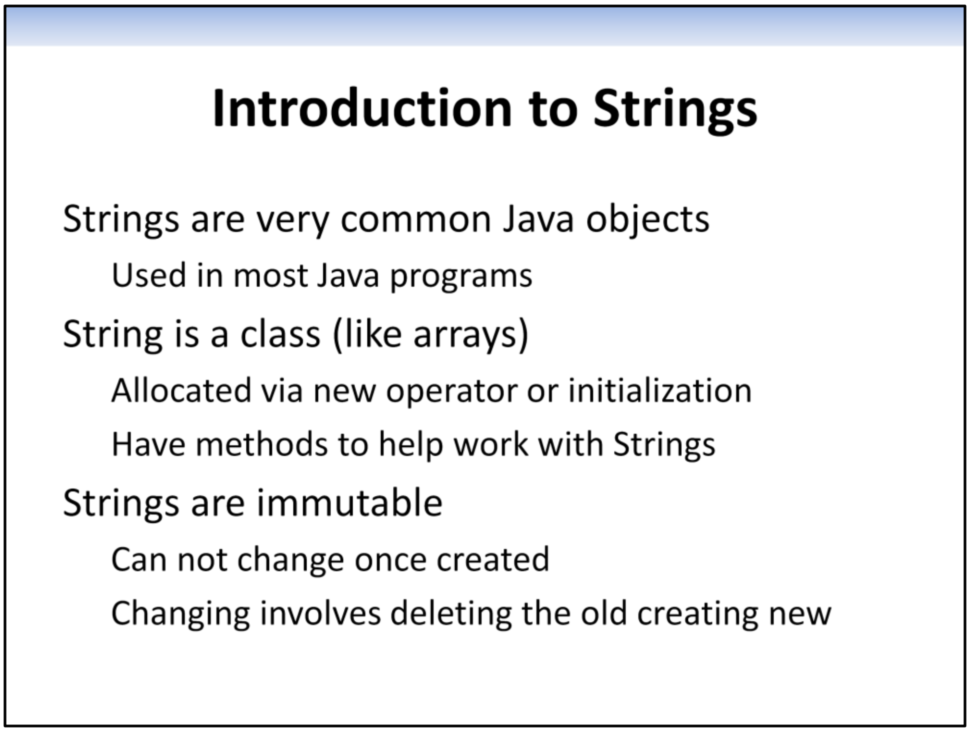
Comparing Strings

Demonstration

The main topics of this presentation are:

- Introduction to Strings
- Creating Strings
- Using Simple Strings
- Comparing Strings
- And a short demonstration illustrating the above concepts

Lets start with an introduction to strings.



Introduction to Strings

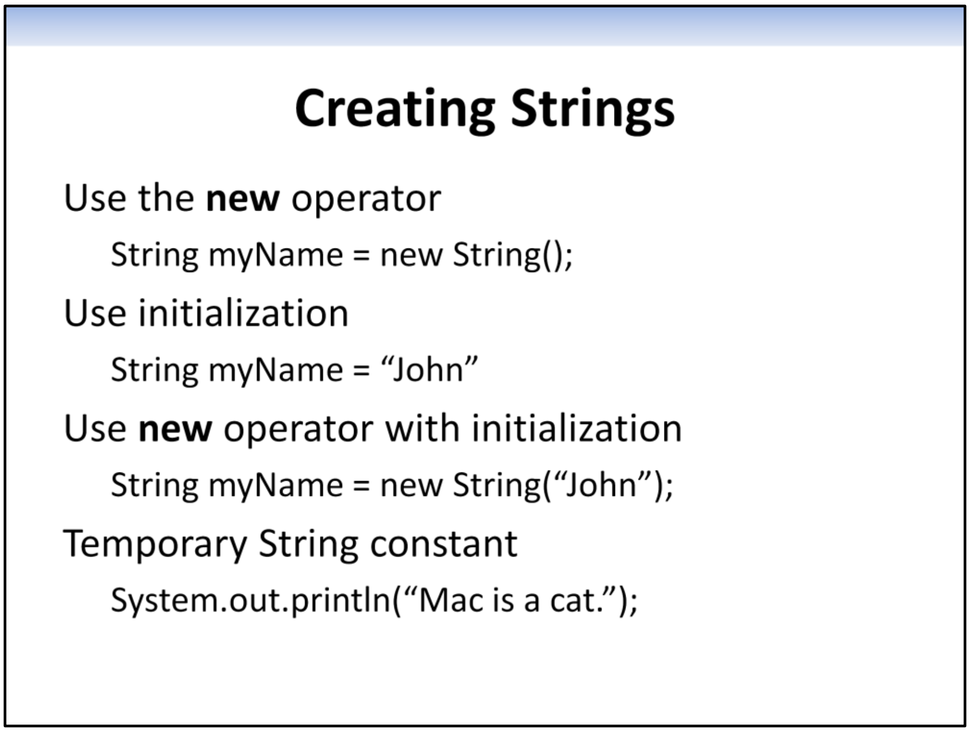
- Strings are very common Java objects
  - Used in most Java programs
- String is a class (like arrays)
  - Allocated via new operator or initialization
  - Have methods to help work with Strings
- Strings are immutable
  - Can not change once created
  - Changing involves deleting the old creating new

Strings are very common in Java programs. They are very convenient in representing text data.   They are basically character arrays with wrapper of methods which provide extensive functionality for dealing with a character array as a single unit. Also there are many methods , for example System.out.println(), which accept Strings as parameters.

It is important to remember strings are ***objects***. Objects are a subject of different presentations but the important part to remember at this point is when dealing with objects you are dealing with references. You are not dealing with the values directly although we will see we can reference the values.

As with all objects, Strings allocate memory and allow for initialization. There are also many methods to make working with Strings relatively convenient.

One very interesting characteristic of Strings is they are immutable, they cannot change once created. This means if you create a string which contains the text “cat” and change it to “cats”, you are actually “deleting” (not really deleting but making it unavailable) the String object “cat” and creating a new String object containing “cats” and replacing the original reference with the reference to this new object.. This process of handling strings has some interesting performance consequences which are visited in a different presentation.



Creating Strings

Use the **new** operator

String myName = new String();

Use initialization

String myName = “John”

Use **new** operator with initialization

String myName = new String(“John”);

Temporary String constant

System.out.println(“Mac is a cat.”);

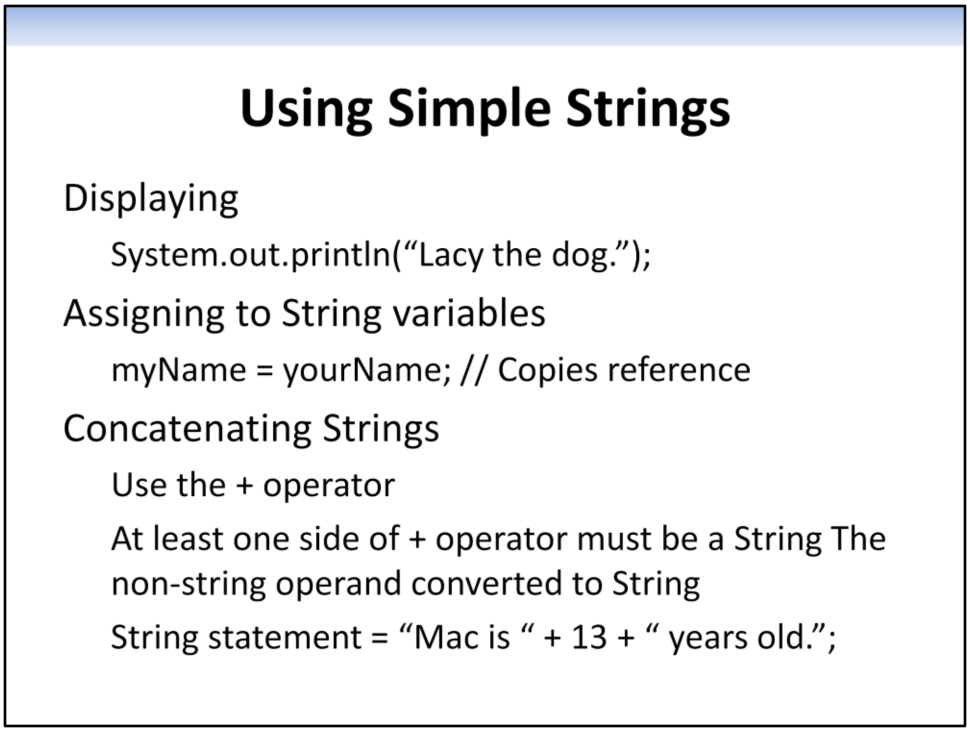
Just like with arrays, you create String objects with the **new** operator. The example here creates a new empty string, An empty string is an object and if displayed would not show anything. Later in the program you could assign a different String object to this String.

Similar to arrays, you can create a string object and initialize it in one step. The example here shows one way to do this. What happens here is the JVM creates a String object which contains the indicated set of characters and assigns the reference to that just created String object to the myName object. The end effect is myName has the value “John”.

Now you can also create and initialize a String using the **new** operator as illustrated here. The effects are almost exactly the same as the earlier initialization example except this initialization is carried out by *copying* the contents of the String parameter, not assigning the reference. This difference can have interesting subtle effects which the later demonstration illustrates.

You also can create temporary strings as shown in the example here. In this case, the string “Mac is a cat.” is created then its reference is passed to the System.out.println() method. Once this method returns the string is de-allocated.

Now lets take a look on how to use strings.



Using Simple Strings

Displaying

System.out.println("Lacy the dog.");

Assigning to String variables

myName = yourName; // Copies reference

Concatenating Strings

Use the + operator

At least one side of + operator must be a String The non-string operand converted to String

String statement = "Mac is " + 13 + " years old.";

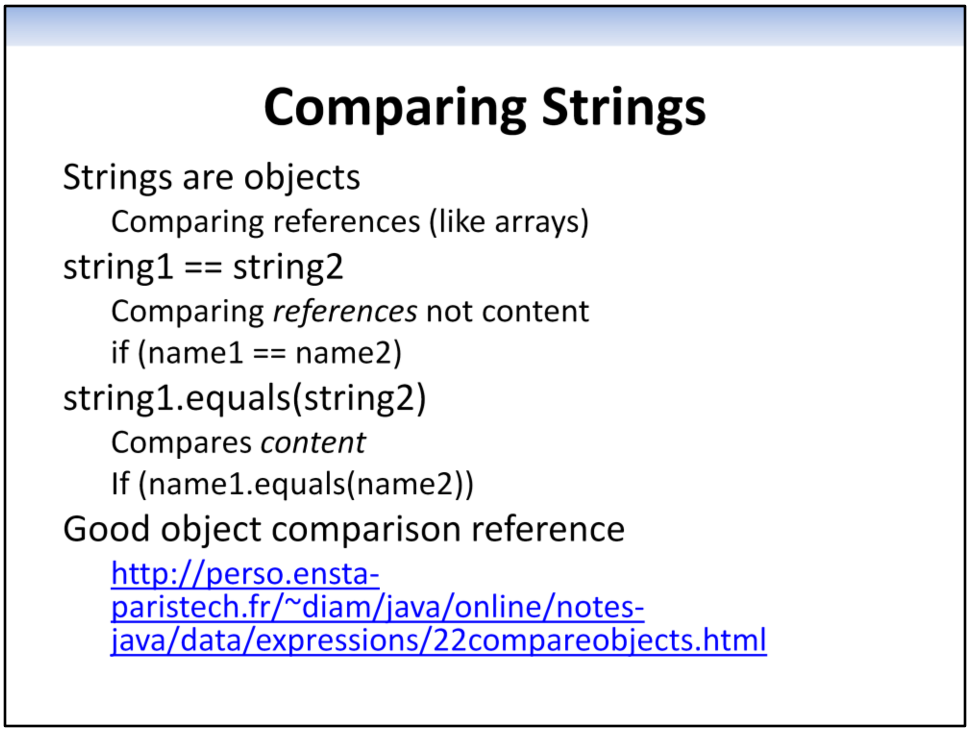
As mentioned previously, strings are very common in Java programs and there are many ways to use them. Lets look at a couple general examples.

You can assign a String to another String. Since Strings are objects, this assignment copies the *reference* of a String to the receiving String. At this point, both myName and yourName refer to the same String object.

You can also concatenate strings by using the **+** operator. This is quite convenient in many situations. To concatenate strings, at least one side of the **+** operator must be a String. The other side can be a primitive type or object which has a **toString()** method (more on objects in a separate presentation.).

In the example here we are taking the string "Mac is ", concatenating the String version of the integer 13 to it, then concatenating the string " years old." to the end. The reference of the String object created by the concatenation is then assigned to the String variable statement.

Next we will look at how to compare Strings.



Comparing Strings

Strings are objects
    Comparing references (like arrays)
string1 == string2
    Comparing *references* not content
    if (name1 == name2)
string1.equals(string2)
    Compares *content*
    If (name1.equals(name2))
Good object comparison reference
    http://perso.ensta-paristech.fr/~diam/java/online/notes-java/data/expressions/22compareobjects.html

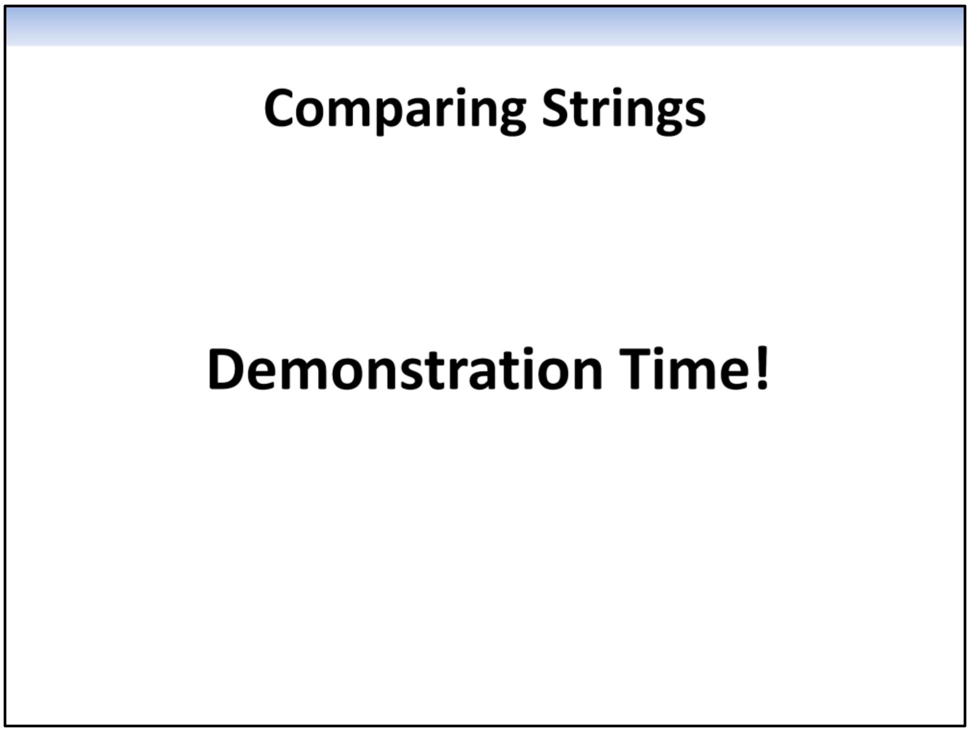
Strings are objects so when comparing Strings it is similar to comparing other objects.

In this first example, we are comparing to see if the *reference* of the String variable string1 is the same *reference* of the String variable string2. If the references are the same true is returned. If they are different false is returned. The contents of string1 and string2 are not examined.

The String class supplies a method **equals()** which does examine the contents of the String variables to determine equality. Even if the objects are different but the contents are the same, a true result is returned.

A good comparison reference is at the URL listed here. At this point, the top two techniques are the most relevant.

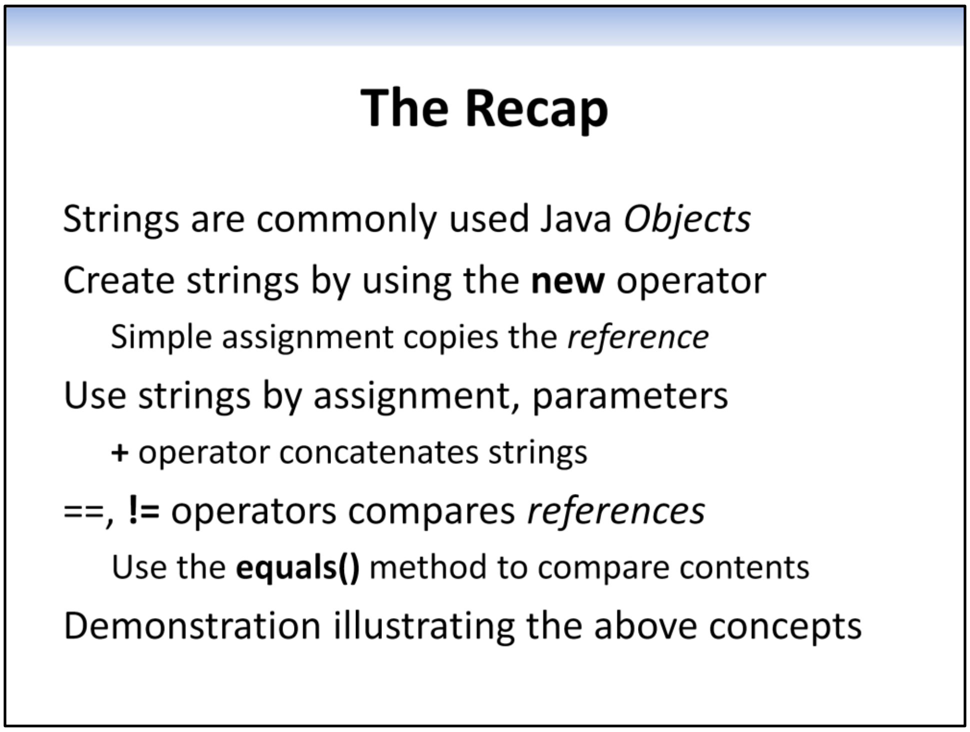
There are several other String class methods which compare strings. More on these in a different presentation.



Comparing Strings

Demonstration Time!

Lets look at a demonstration which illustrates many of the concepts discussed so far in this presentation.



The Recap

Strings are commonly used Java *Objects*

Create strings by using the **new** operator

- Simple assignment copies the *reference*

Use strings by assignment, parameters

- **+** operator concatenates strings

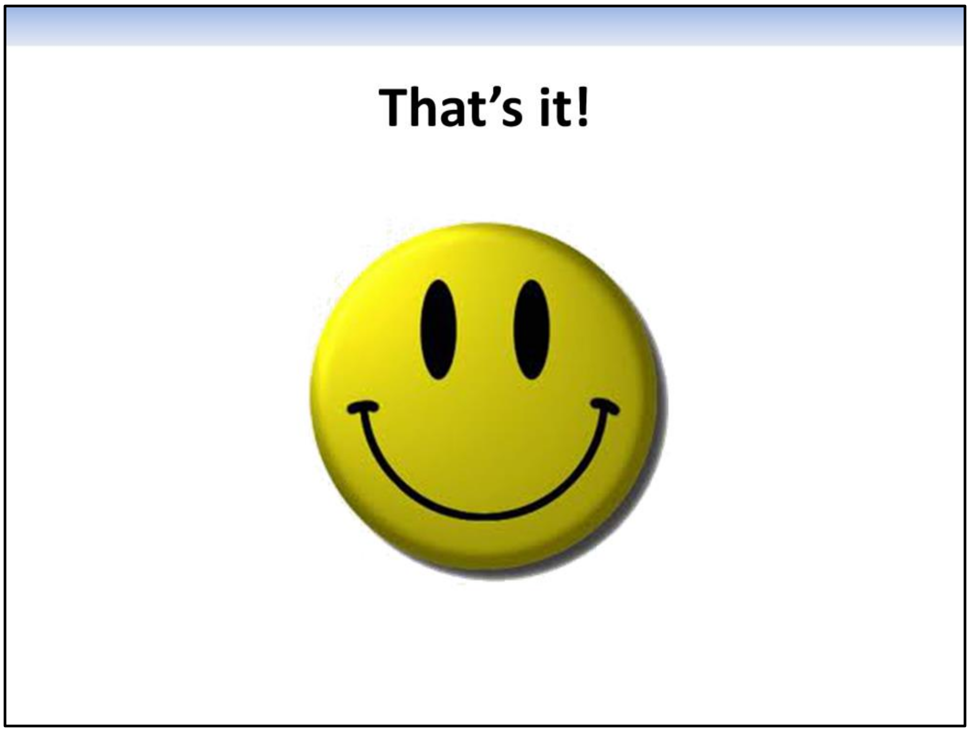
==, **!=** operators compares *references*

- Use the **equals()** method to compare contents

Demonstration illustrating the above concepts

OK here is the recap of the main points of this presentation:

- Strings are very commonly used Java objects. They are objects, not a primitive type.
- You can create strings by using the **new** operator. You can also create Strings with initialization by assigning a String constant at creation time. You can also provide initialization values when using the **new** operator.
- You can use strings in assignments and as parameters to methods. One very common technique is to concatenate strings with non-string objects or primitive types to create new Strings.
- When comparing strings for equality, the **==** operator determines if the two String object *references* are the same. The contents are not examined. If you want to determine if the *contents* of the two String objects are the same, you use the **equals()** method.
- We ended with a demonstration which illustrated the above concepts.



That’s it!

That’s it!