

Object-Oriented Programming: Using Classes

- Class basics and benefits.
- Creating objects using constructors.
- Calling Methods.
- Using predefined Java classes.

Class Basics and Benefits:

- A *class* combines:
 - Data — identifiers that hold values. Can be any type (**int**, **float**, String, etc.)
 - Methods — code that manipulates the data.
- Classes are a template (or blueprint) used to create specific objects.
- All Java programs consist of at least one class.
- Example:
 - *BankAccount* class:
 - Data: name of account holder, account number, balance, mailing address, ...
 - Methods: set or get the value of each piece of data, compute new balance after a deposit or withdrawal
 - A specific instance of *BankAccount* might be an object called **myDreamAccount**.
 - Data: **Eric; 024874898; \$21,698,278.42; ...**

Class Basics and Benefits (continued):

Terminology:

- A class is declared exactly once in a program.
 - The declaration of a class can also be done in a library. The **Scanner** and **String** classes are examples.
- **Instances** of the class can be created. There can be many of these (analogous to having many **int**'s).

- **Object reference:** the identifier of the object. e.g.

```
String myName, yourName;
```

```
BankAccount yours, mine, ours, ourKids;
```

- The identifiers **myName, yourName, yours, mine, ours, ourKids** above are object references. They can refer (point) to a instance of an object.

Reference Variables:

- A variable which has a primitive type contains a value of that type.

```
int i = 10;           // i contains the value 10
```

- A variable that is declared to have a class type is a reference (pointer) to an object.

```
String s;           // s contains a reference to a String object
```

- Just declaring a variable does not create an object. Reference values are initialized with the value **null**, indicating they don't refer to anything yet.

i	10
s	null

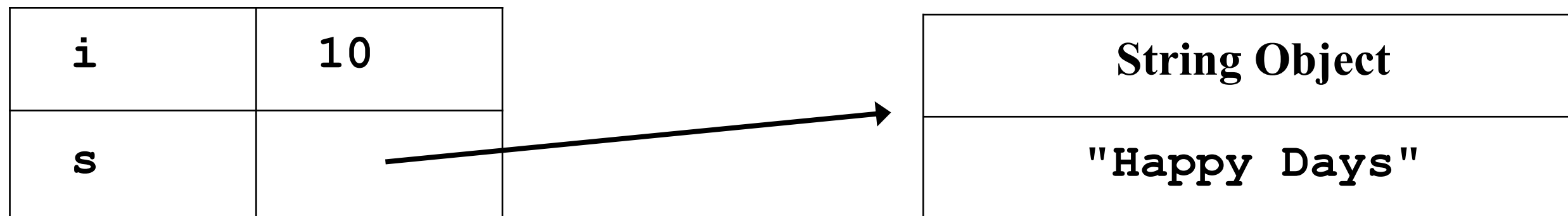
Instantiating an object: creating an object of a class.

- You can instantiate an object by using the keyword **new**:

```
int i = 10;
```

```
String s;
```

```
s = new String("Happy Days");
```



More Terminology:

- **Instance of the class:** an object.
- **Methods:** the code to manipulate the object data.

```
int xray = scanInput.nextInt();
```

```
// nextInt is a method of the Scanner class
```

- **Constructor:** special method that creates an object and assigns initial values to the data.

```
Scanner myInput;
```

```
myInput = new Scanner( System.in );
```

```
// uses the constructor to create a Scanner object
```

More Terminology (continued):

- **Calling a method:** invoking the code to perform a service for an object.

```
String nextWord, anotherWord;
```

```
anotherWord = new String("Bibble");
```

```
shortWord = anotherWord.substring( 0, 3 );
```

```
// an invocation of the substring method
```

```
nextWord = myInput.next(); // an invocation of the next method.
```

Naming Conventions:

- Class names: start with an upper-case letter.
 - Capitalize internal words.
 - Examples:

Scanner

String

BankAccount

HomeAddress

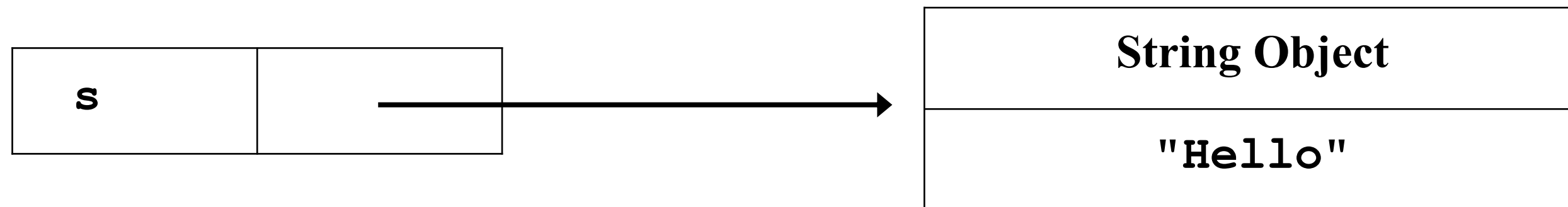
Strings:

- Strings are objects, not a primitive type (the class is String).
- In one way strings are unique as objects since they have a literal representation.
 - Therefore you can write:

```
String s = "Hello";
```

- As well as:

```
String s = new String("Hello");
```



Strings - Concatenation:

- We've already seen that the `+` operator concatenates strings.
- If one of the operands of the `+` operator is a string, the other will be converted to a string if necessary.

```
3 + "Hello" // produces the string "3Hello"
```

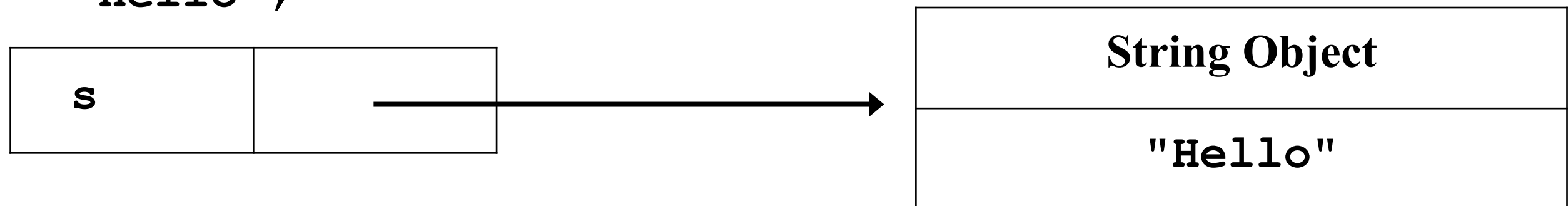
```
1 + 2 + "Hello" // produces "3Hello"
```

```
"Hello" + 1 + 2 // produces "Hello12"
```

Strings:

- Strings are *immutable* objects which means their value cannot change.
- Does this mean you can't reassign a string variable?
- No, the reference changes but the object does not.

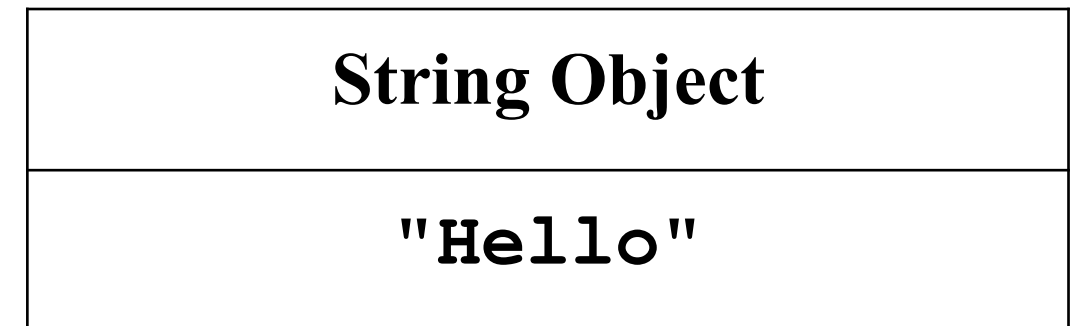
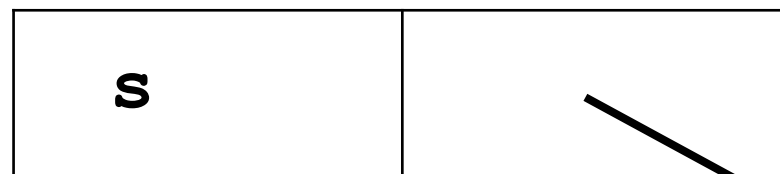
```
String s = "Hello";
```



Strings:

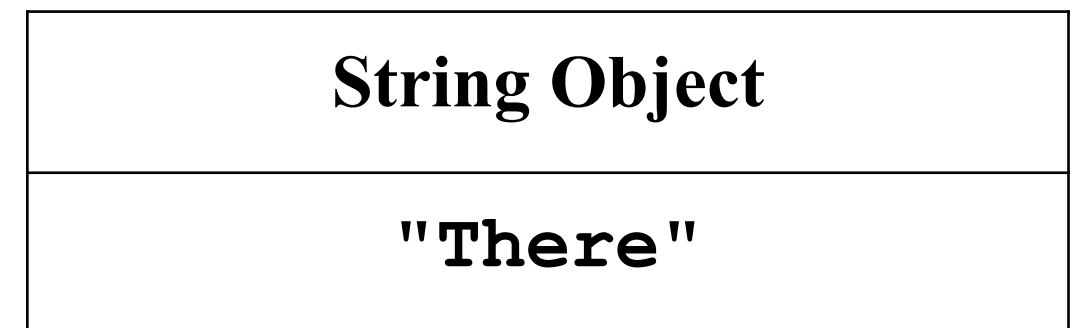
- Strings are *immutable* objects which means their value cannot change.
- Does this mean you can't reassign a string variable?
- No, the reference changes but the object does not.

```
String s = "Hello";
```



```
s = "There";
```

- What happens to the first string object?



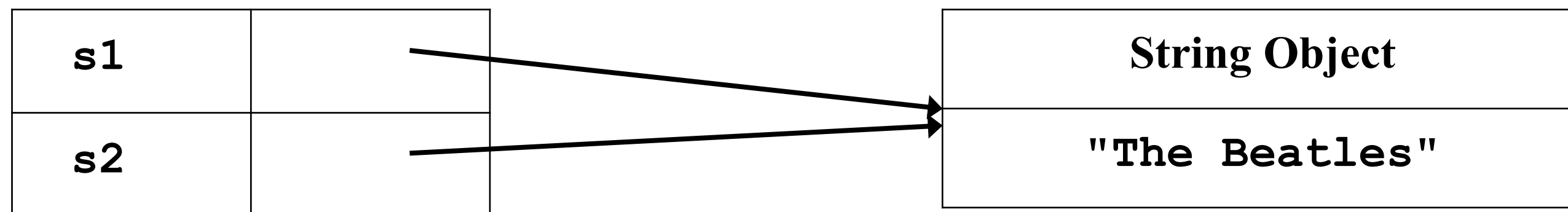
Comparing Strings:

- Suppose we have.

```
String s1 = "The Beatles";
```

```
String s2 = s1;
```

- Does `s1 == s2` evaluate to **true** or **false**?
- **true**



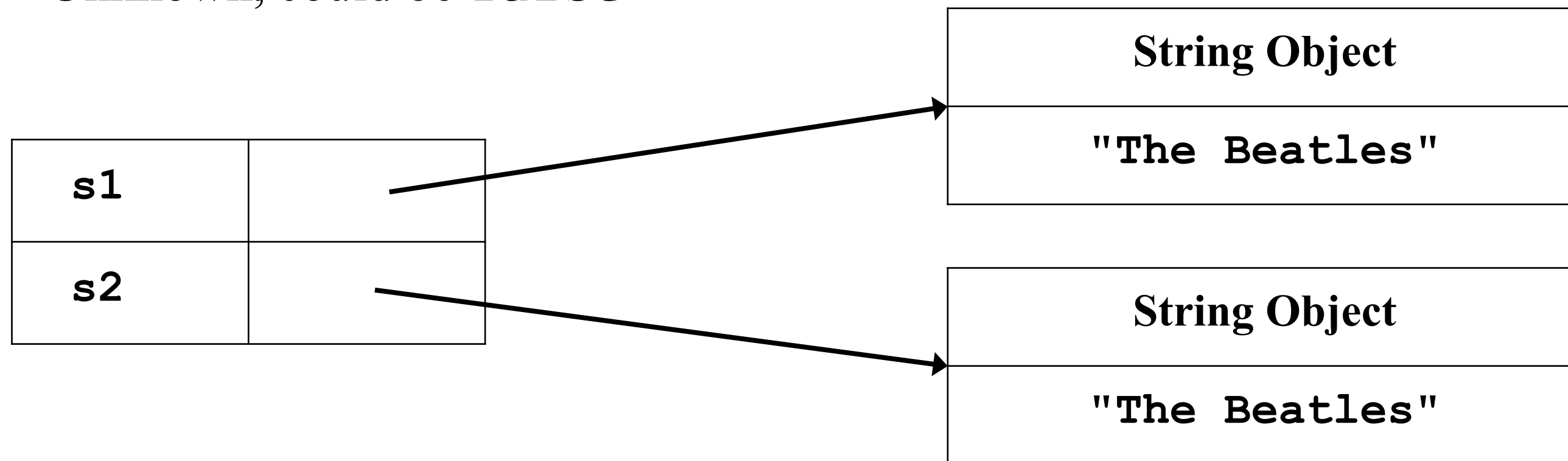
Comparing Strings:

- But what about?

```
String s1 = "The Beatles";
```

```
String s2 = "The Beatles";
```

- Does `s1 == s2` evaluate to **true** or **false**?
- Unknown, could be **false**



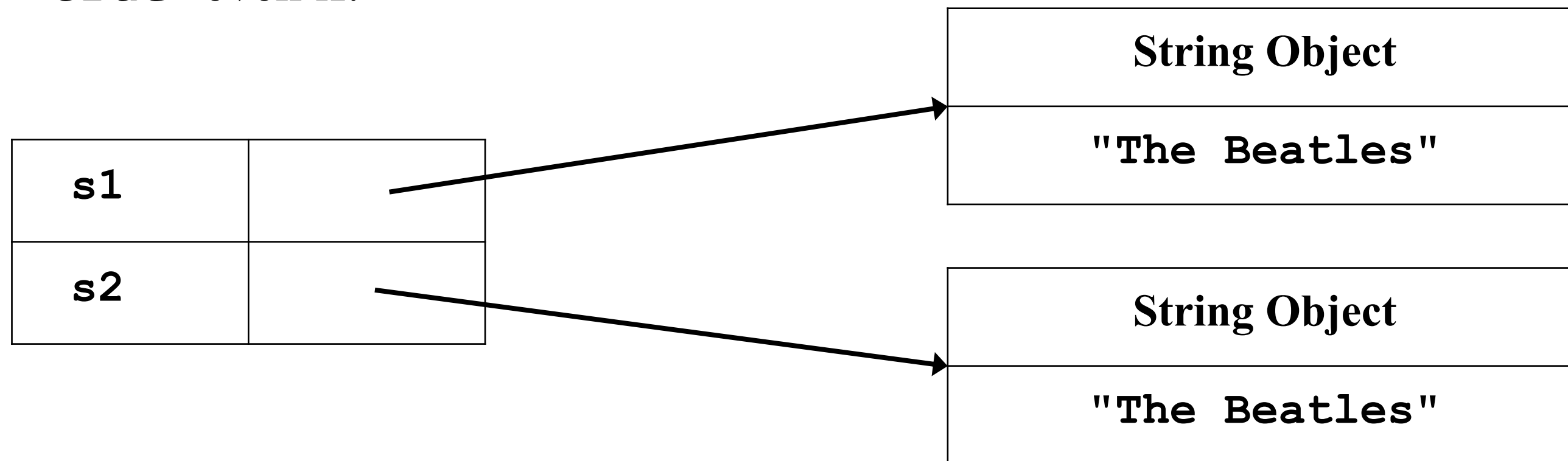
Comparing Strings:

- When comparing strings, usually want to use the **equals** method.

```
String s1 = "The Beatles";
```

```
String s2 = "The Beatles";
```

- Does **s1.equals(s2)** evaluate to **true** or **false**?
- **true** even if:



String Methods:

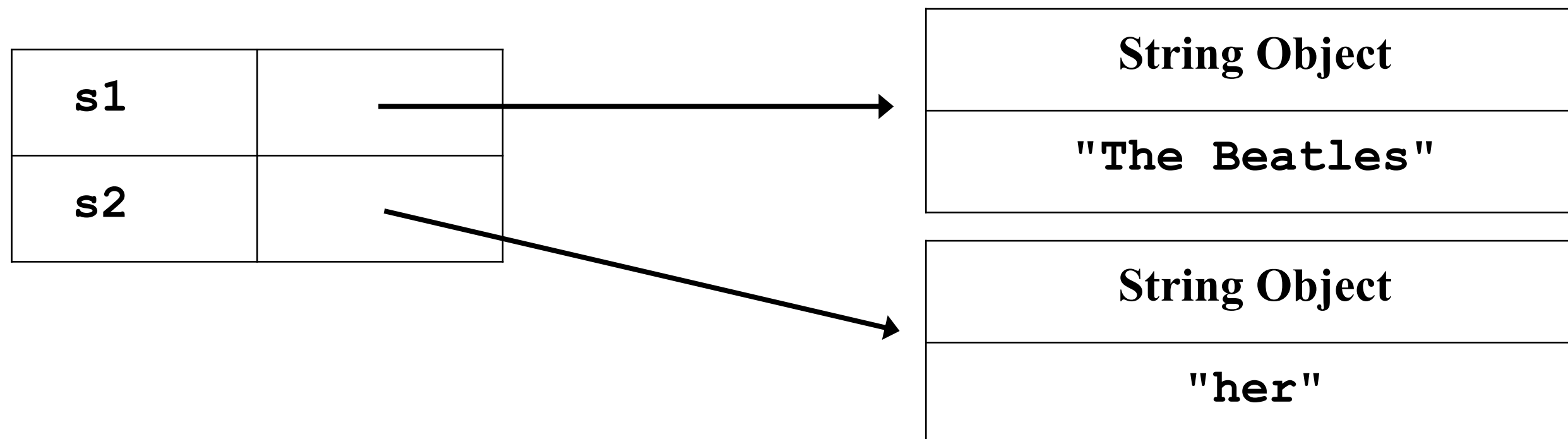
- Strings have many methods including **length** and **substring**.

```
String s1 = "The Beatles";
```

```
int len = s1.length(); //stores 11 into len
```

```
String s2 = s1.substring(1,3); // creates a new String object
```

```
// with value "he" and points s2 to it
```



String Methods

- Getting a character from the String:

```
char aLetter;
```

```
String myName = "Charlie Brown";
```

```
aLetter = myName.charAt(4);
```

- The **charAt** method *returns* a **char**; thus, **aLetter** is of type **char**
- Strings are 0 indexed, so the above stores 'l' in aLetter

String Methods:

- The **indexOf** method here returns an **int** that is the index of the first occurrence of a character:

```
public class IndexOfExampleOne {
    public static void main(String[] args) {
        String myName = "Charlie Brown";
        String restOfName;
        int location;
        char aLetter = 'e';

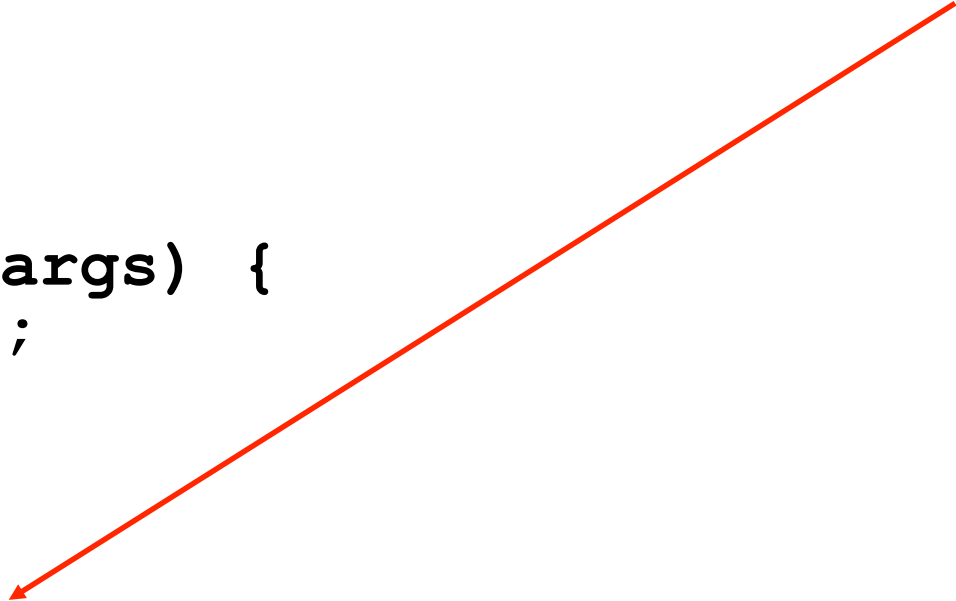
        location = myName.indexOf( aLetter );

        System.out.println(aLetter + " first appears at position " +
                           location);

        restOfName = myName.substring( location + 1, myName.length() );

        System.out.println("The rest of the name is '" + restOfName + "'");

    } // end of main method
} // end of class IndexOfExampleOne
```



String Methods

- There are two versions of **indexOf**. The second version finds the first occurrence of a **String**.

```
public class IndexOfExampleTwo {
    public static void main(String[] args) {
        String myName = "Charlie Brown";
        String restOfName;
        int location;
        String lookFor = "lie";

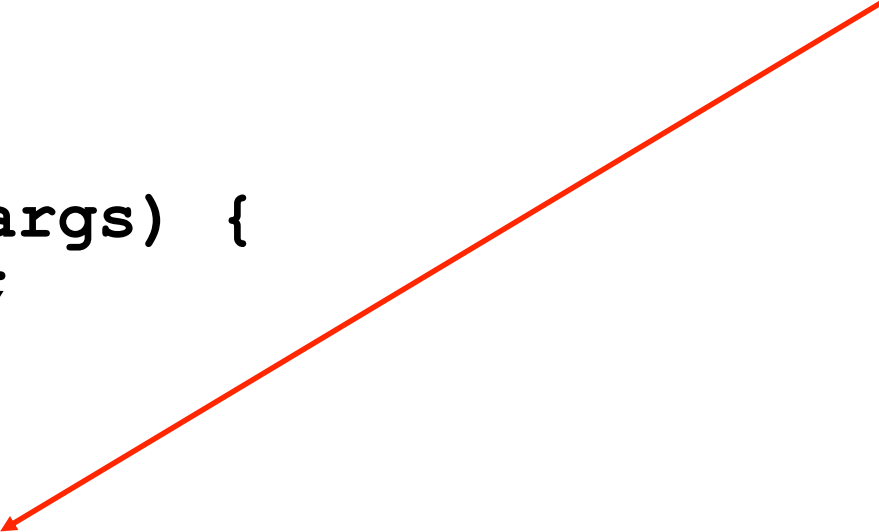
        location = myName.indexOf( lookFor );

        System.out.println( lookFor + " starts at position " + location);

        restOfName = myName.substring( location, myName.length() );

        System.out.println("The rest of the name is '" + restOfName + "'");

    } // end of main method
} // end of class IndexOfExampleTwo
```

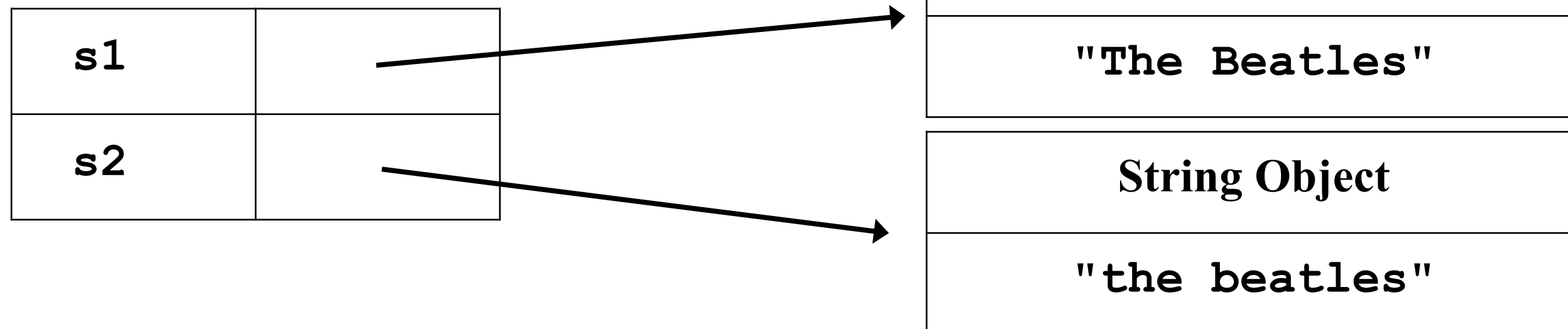


String Methods:

- The **toLowerCase** method returns a new **String** containing the original characters, but with all letters in lower case.
- The **toUpperCase** method is similar to **toLowerCase**, but returns the original string with all letters in upper case.

```
String s1 = "The Beatles";
```

```
String s2 = s1.toLowerCase();
```



String Methods:

- Summary: **String** methods covered
 - `length()`
 - `charAt(int)`
 - `indexOf(char)`
 - `indexOf(String)`
 - `substring(int, int)`
 - `toLowerCase()`
 - `toUpperCase()`
- See the Java API for all the String methods.

Class Scanner:

- How do we (“us humans”) tell a program something?

- Example: A program that will print my name:

```
String myName = "Eric";
```

```
System.out.println("The human's name is " + myName);
```

- But, someone not named Eric might want to use the program; hmmm...?

Class Scanner:

- The *Scanner* class allows a program to read input from the keyboard.
- Three steps:
 1. Tell Java that the *Scanner* class will be used:

```
import java.util.Scanner;
```

2. Declare an instance of the *Scanner* class and connect it to the keyboard:

```
Scanner scanInput;
```

```
scanInput = new Scanner( System.in );
```

3. Read a value from the keyboard:

```
String myName = scanInput.next();
```

Class Scanner (continued):

- The first step tells Java that the program will use the *Scanner* class.
 - The *Scanner* class is part of a package of classes known as **java.util**
 - Put the **import** line before the declaration of your class:

```
import java.util.Scanner;  
public class ScannerSample  
{ ...
```

Two ways to do this:

To get just the one class:

```
import java.util.Scanner;
```

To get to all classes in the **java.util** package:

```
import java.util.*;
```


Class Scanner (continued):

- The second step declares an instance of the *Scanner* class inside **main**:

```
public static void main( String [] args )
{
    Scanner scanInput;
    scanInput = new Scanner( System.in );
}
```

- To give **scanInput** a value, we must create an instance of *Scanner*. The reserved word **new** is used for this.
- To have this instance of *Scanner* be connected to the keyboard, we use **System.in**.
 - Instances of *Scanner* can be connected to other “things”, such as files.
 - For now, we will be using only **System.in**.

Class Scanner (continued):

- What we have so far:

```
import java.util.Scanner;
public class ScannerSample
{
    public static void main(String[] args)
    {
        Scanner scanInput;
        scanInput = new Scanner( System.in );
    }
}
```

Class Scanner (continued):

- The third step is to get the user's input from the keyboard.

- First, we ask the user:

```
System.out.print("Enter your name: ");
```

- Note: use of **print**, not **println**. (What is the difference?)

- Then, we get the answer:

```
String myName = scanInput.next();
```

- Classes provide *methods* that are used to extract data values from the class.
- The **next()** method of the *Scanner* class will read the next string the user types.
- Declare a string, **myName**, to hold the answer returned by the **next()** method.

- A complete program.

```
import java.util.Scanner;

public class ScannerSample {
    public static void main(String[] args) {
        Scanner scanInput;
        scanInput = new Scanner( System.in );

        System.out.print("Enter your name: ");
        String myName = scanInput.next();

        System.out.println("myName is " + myName);

    } // end of main method
} // end of class ScannerSample
```

- Questions about program on previous slide:
 - What happens if the user types just one character?
 - What happens if the user types a really long name?
 - What happens if the user types characters other than letters?
 - What happens if the user types two (or more) names?

Class Scanner (continued):

- How does *Scanner*'s **next ()** method “know” where the **String** ends?
 - When the user presses return or enter.
 - When a blank space is found.
 - When a tab is found.
 - Collectively, these (blank space, tab, newline) are known as *whitespace* characters.

Class Scanner (continued):

- One Scanner, many uses.

```
import java.util.Scanner;
```

```
public class ScannerSampleAgain {
    public static void main(String[] args)
    {
        Scanner scanInput;
        scanInput = new Scanner( System.in );

        System.out.print("Enter your name: ");
        String myName = scanInput.next();

        System.out.println("myName is " + myName);

        System.out.print("Enter another name: ");
        myName = scanInput.next();

        System.out.println("myName is " + myName);

    } // end of main method
} // end of class ScannerSampleAgain
```

Declare Scanner once.

Create the Scanner once.

Use the (same) Scanner over and over.

Class Scanner (continued):

- In general, methods within a class are used to:
 - Get the value of a data item contained in the class.
 - Set the value of a data item in the class.
 - Perform a calculation that, in part, involves data item(s) in the class.

- Methods that get the value of a data item have a *return type*.
 - The **next ()** method of **Scanner** that was used on the previous slide, returns a **String**.
 - Other useful **Scanner** examples include:
 - **String nextLine ()**
 - Returns a **String** that contains all the characters typed up to the next newline.
 - Provides a way to get blank spaces and tab characters included in the returned **String**.

Class Scanner (continued):

- **int nextInt()**
 - Returns an **int** that contains the next integer typed on the keyboard.
 - Whitespace before the integer will be skipped.
 - What happens if:
 - The user types a number that contains a decimal point? a comma?
 - The user types several numbers on the same line?

Class Scanner (continued):

- Example on how to scan an integer.

```
import java.util.Scanner;

public class ScannerInt {
    public static void main(String[] args) {
        Scanner scanInput;
        scanInput = new Scanner( System.in );

        System.out.print("Enter an integer: ");
        int myNumber = scanInput.nextInt();

        System.out.println("myNumber is " + myNumber);
    } // end of main method
} // end of class ScannerInt
```

Class Scanner (continued):

- **double nextDouble ()**
 - Returns a **double** that contains the next **double** typed on the keyboard.
 - Whitespace before the number will be skipped.
 - What happens if:
 - The user types a number that contains a decimal point? a comma?
 - The user types several numbers on the same line?

- Example on how to scan an **double**.

```
import java.util.Scanner;

public class ScannerDouble {
    public static void main(String[] args) {
        Scanner scanInput;
        scanInput = new Scanner( System.in );

        System.out.print("Enter a double: ");
        double myNumber = scanInput.nextDouble();

        System.out.println("myNumber is " + myNumber);
    } // end of main method
} // end of class ScannerDouble
```

Return value	Method name	Description
<code>byte</code>	<code>nextByte ()</code>	returns the next input as a byte
<code>short</code>	<code>nextShort ()</code>	returns the next input as a short
<code>int</code>	<code>nextInt ()</code>	returns the next input as an int
<code>long</code>	<code>nextLong ()</code>	returns the next input as a long
<code>float</code>	<code>nextFloat ()</code>	returns the next input as a float
<code>double</code>	<code>nextDouble ()</code>	returns the next input as a double
<code>boolean</code>	<code>nextBoolean ()</code>	returns the next input as a boolean
<code>String</code>	<code>next ()</code>	returns the next token in the input line as a String
<code>String</code>	<code>nextLine ()</code>	returns the input line as a String

- See also the Java API for `java.util.Scanner`