Lecture 9

- Covers
  - The String class
- Reading: Savitch 2.2

Lecture overview

- String as objects of the String class
- String concatenation
- String declaration and instantiation
- String immutability
- String length and indexes
- String comparison
- Common string operations

- Objects of the String class

Strings

- Sequence of characters
- Treated as a single entity
- Not a primitive type in Java
- Instances of the String class

String literals

- Objects of the String class
- Constant values, cannot be changed
- Written between double quotes ""
  - "Trust no one"
  - System.out.println("Trust no 1");

Escape sequences

- Can be included inside a string and are treated as a single character
- Begin with the \ symbol
- Some of the most used escape sequences

\n	newline
\t	tab
\\	backslash (\)
\'	single quote
\"	double quote

Examples

```
System.out.println("They\'re  Watching");

System.out.println("\"Babylon  5\" is a space  station");

System.out.println("I\nam\na\nfish");
```

Unicode characters

- You can include the Unicode value of any character in a string
- Give the 4-digit hexadecimal Unicode value following \u
- A Unicode character is treated as a single character within the string

Example

```
System.out.println("The truth is out there");

System.out.println("\'\u00E9\u00ED " +
        "\'aan\u00ED\u00EDg\u00D3\u00D3 " +
        "\'\u00E1hoot\'\u00E9");
```

→ 'éí 'aaníígÓÓ 'áhoot'é

```
System.out.println("Die Wahrheit Ist " +
                   "Irgendwo Da Drau\u03B2en");
```

→ Die Wahrheit Ist Irgendwo Da Drau?en

Sometimes the computer does not support all parts of the Unicode character set. This should have been: β

- String concatenation

String concatenation

- Concatenation operator + creates a new string made up of the lhs concatenated with the rhs

```
System.out.println("Deny" + " Everything");
System.out.println("Nothing Important" +
                   " Happened Today");
System.out.println("Believe" +
                   " the" + " lie");
```

Concatenation with numbers

- When a number is concatenated with a String the result is a String

```
System.out.println("The meaning of life is " + 42);
System.out.println("The meaning of life is " + "42");

int meaningOfLife = 42;
System.out.println("The meaning of life is " +
                                    meaningOfLife);
int x = 4, y = 2;
System.out.println("The meaning of life is " + x + y);
```

Concatenation with numbers (caution)

```
int x = 4, y = 2;

System.out.println("The meaning of life is " + x + y);

System.out.println("The meaning of life is " + (x + y));

System.out.println(x + y + ", the meaning of life is");

System.out.println(x + (y + ", the meaning of life is"));
```

Concatenation with numbers (caution)

```
int cost = 10;
int profit = 2;
System.out.println("price is " + cost + profit);
```

⇨ displays: price is 102

```
System.out.println("price is " + (cost + profit));
```

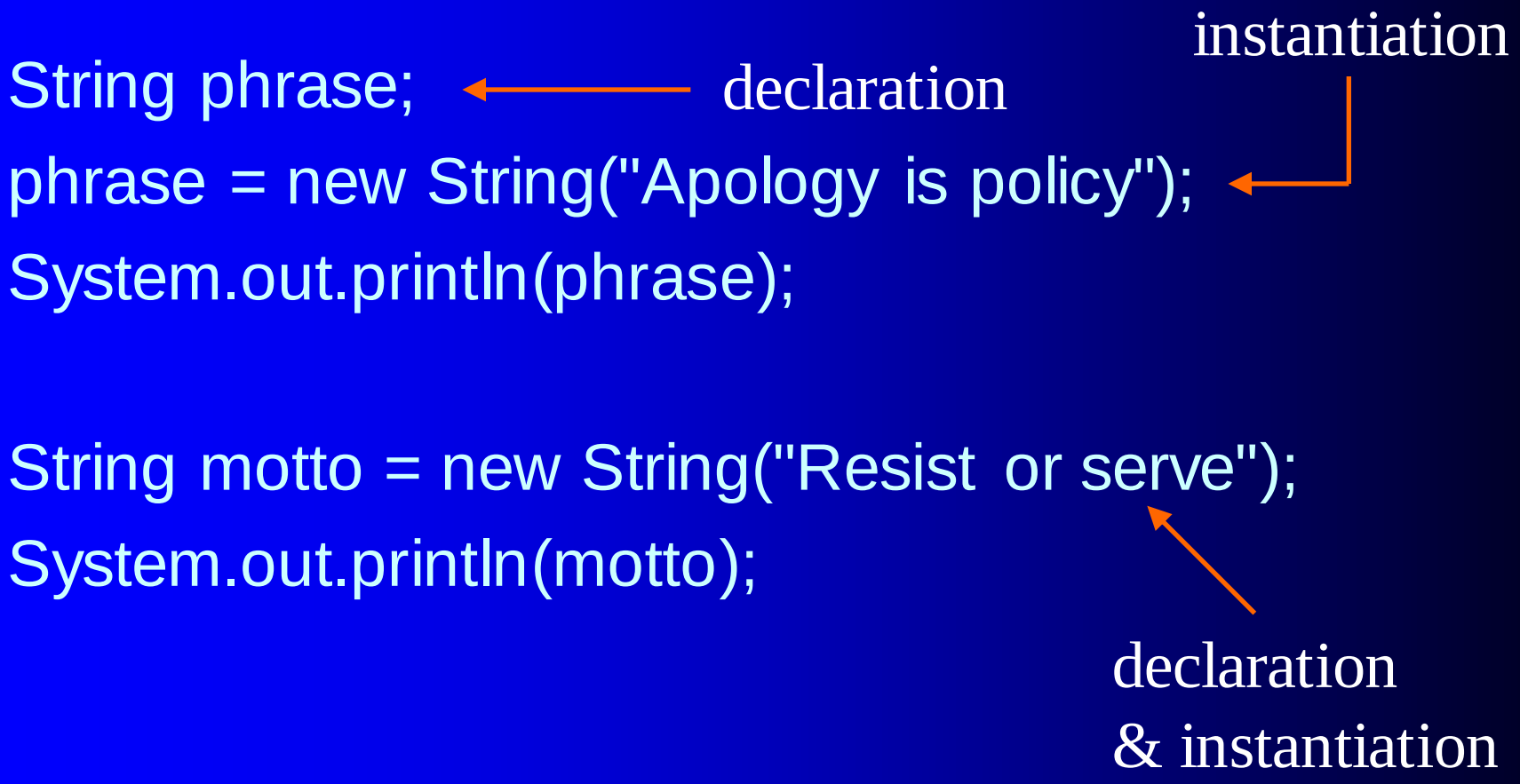
⇨ displays: price is 12

- String declaration and instantiation

Objects & variables of the String class

- Each string literal is an object of the String class
  - It has a state (the character sequence it holds)
  - It has behaviour (the operations it can carry out on the string)
- We can create variables of the String class (we do this by "declaration statements")

String declarations (and instantiations)



String instantiation

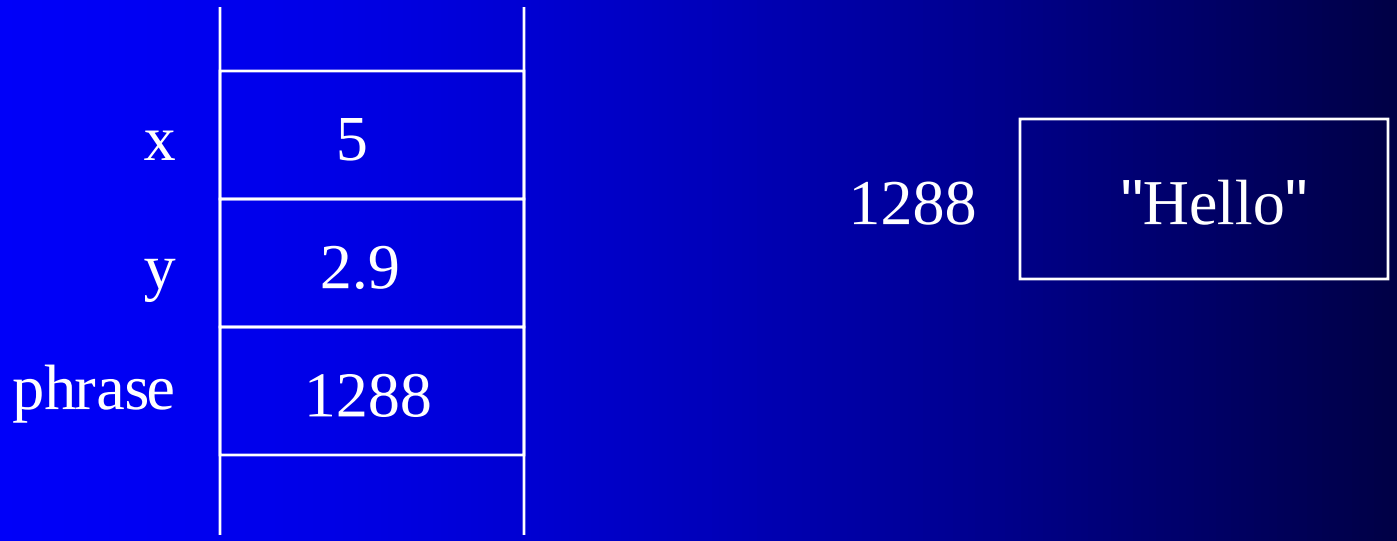
- Unlike declarations for primitive types, a declaration of a variable with a class type does not reserve space in memory to store the object - *it creates enough space to store the address of an object*
- We must tell the compiler if we want it to create a new object (and reserve memory to store it) with the new( ) operation

Variables and references

- A variable is either a variable of primitive type or a variable of class type.
- A variable of primitive type stores the actual value
- A variable of class type stores the address of an object.
  - We call the variable a reference to the object
  - A declaration of a variable with a class type does not reserve space in memory to store the object - it creates enough space to store the address of an object

String instantiation

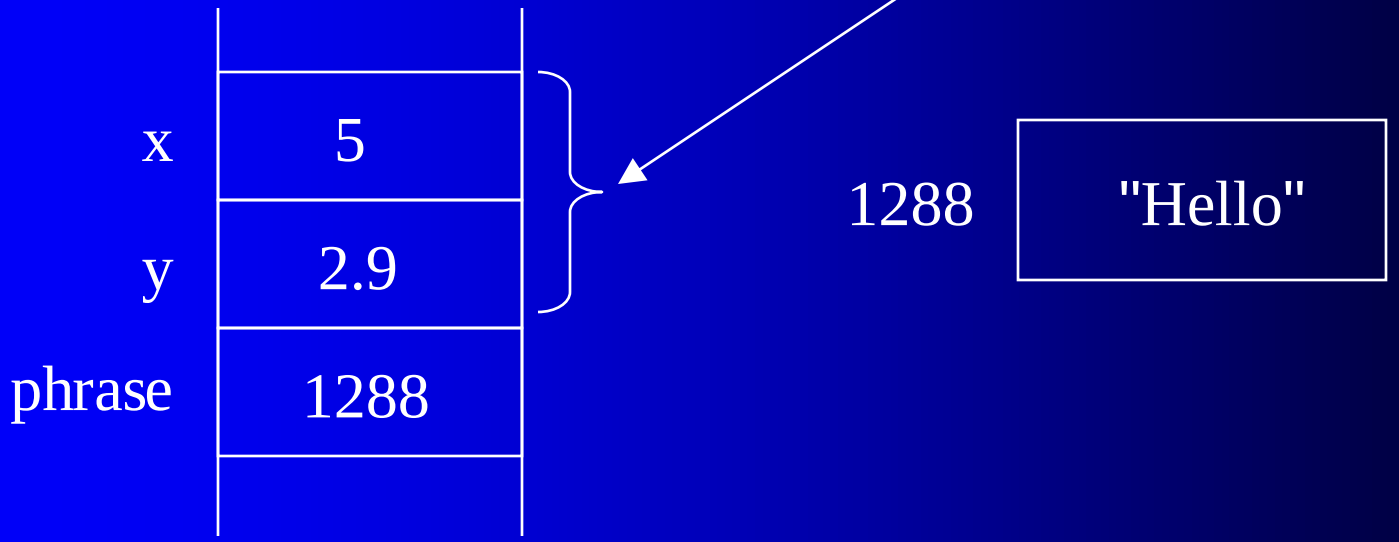
```
int x = 5;
double y = 2.9;
String phrase = new String("Hello");
```



String instantiation

```
int x = 5;
double y = 2.9;
String phrase = new String("Hello");
```

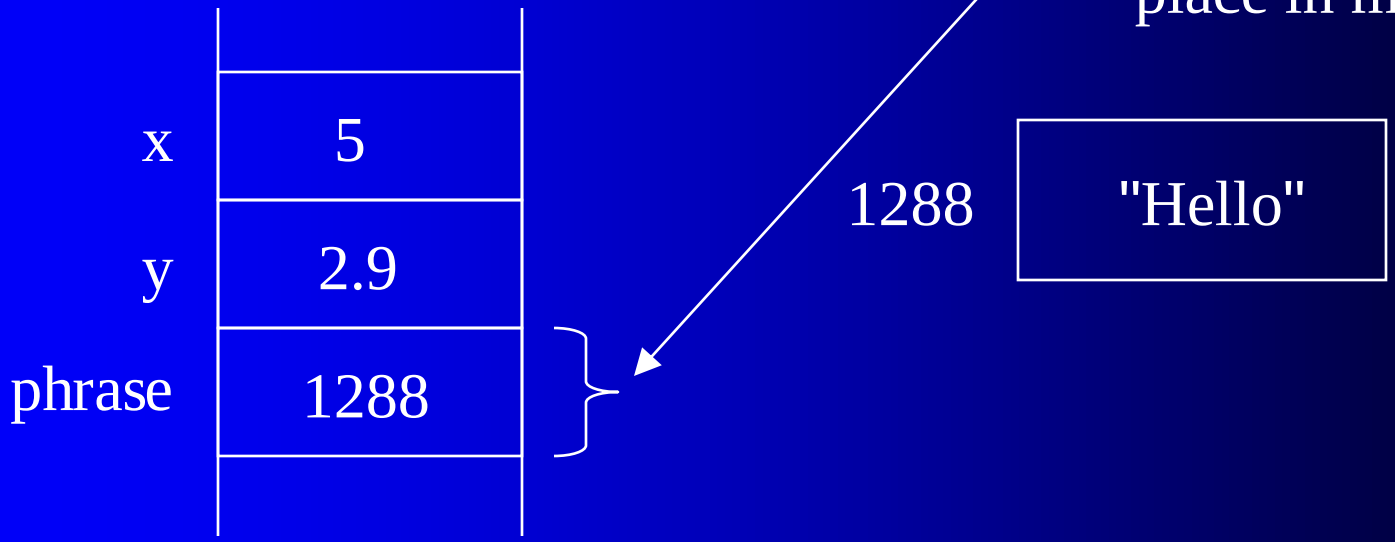
variables of primitive types store the actual value



String instantiation

```
int x = 5;
double y = 2.9;
String phrase = new String("Hello");
```

variables of String type store the location (memory address) of a String object which is stored in a different place in memory



String instantiation

String motto;

– Gives us a name by which to refer to a string

motto = new String("Trust me");

– Creates a new string which contains the characters "Trust me" and then makes the variable motto refer to that string

– We call the variable a reference to the string

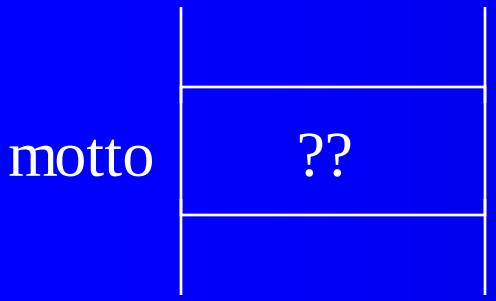
Constructors

- When we create a new String object we want it to have an initial value
- The constructor of the String class sets the initial value of the String object to the value we give the new( ) operation

```
String motto = new String("I want to believe");
```

String instantiation

```
String motto = new String("I want to believe");
```



String instantiation

```
String motto = new String("I want to believe");
```

String instantiation

```
String motto = new String("I want to believe");
```

Alternate construction of a String

- In general, to create a new object of any class type we must use the new( ) operation
- Strings are one of the most commonly used objects and a shorthand notation exists for creating String objects

```
String motto = "I want to believe";
```

- String immutability

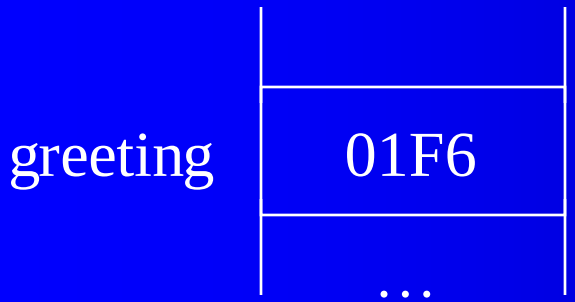
String immutability

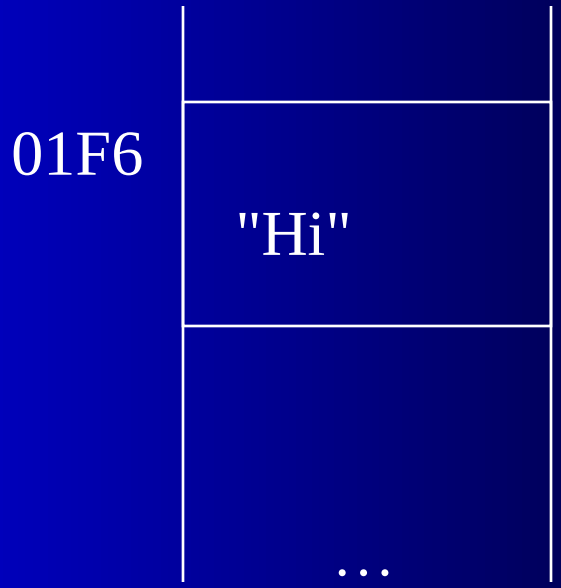
- Strings cannot change their contents
- This property is called immutability
- When we make a String variable refer to another String object we are not changing the content of the object it refers to, but changing which object it refers to

```
String greeting = "Hi";
greeting = "Hi" + " There";
greeting = greeting + " Class!";
```

Example

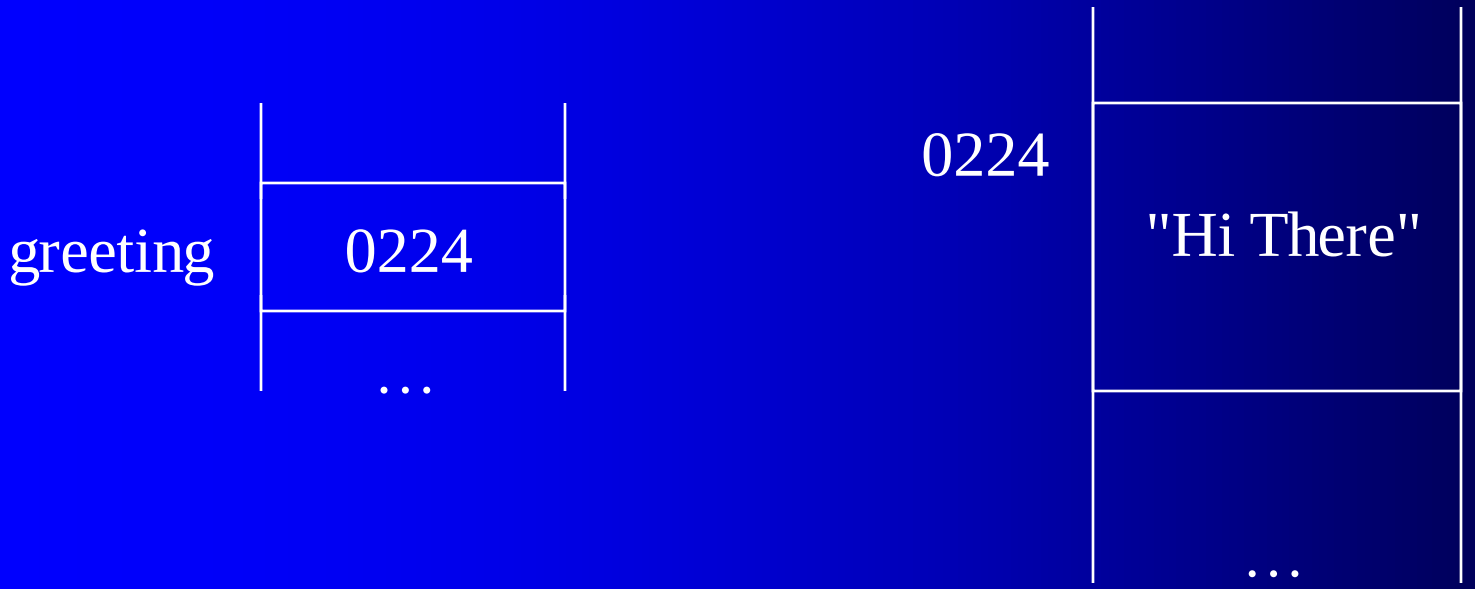
```
String greeting = "Hi";
greeting = "Hi" + " There";
greeting = greeting + " Class!";
```





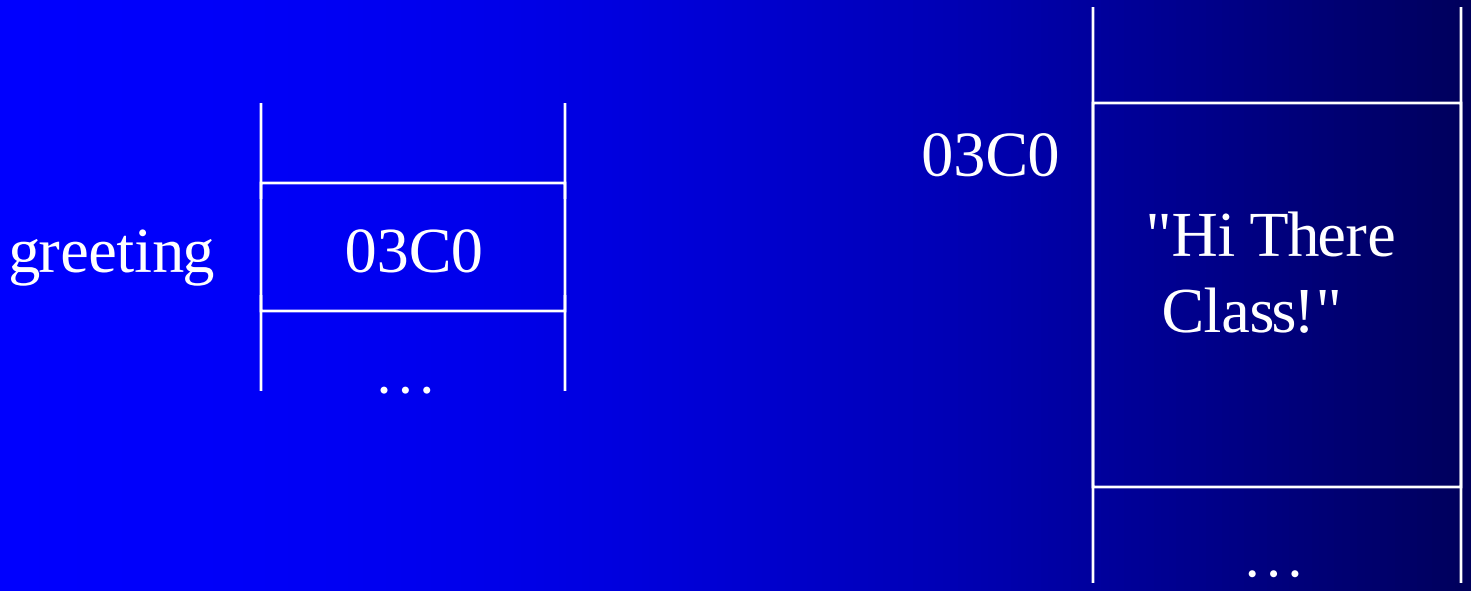
Example

```
String greeting = "Hi";
greeting = "Hi" + " There";
greeting = greeting + " Class!";
```



Example

```
String greeting = "Hi";
greeting = "Hi" + " There";
greeting = greeting + " Class!";
```



- String length and indexes

The dot operator

- The dot (.) operator allows us to access the operations of an object or class
- A class example

  Math.pow(x,y)

  – Accesses the class method pow( ) in the class Math
- The dot operator can access the methods of object variables or literals

  "Resist or serve".length( )

  motto.length( )

String length

- The length( ) operation of the String class is an instance method that tells us the number of characters in that String object

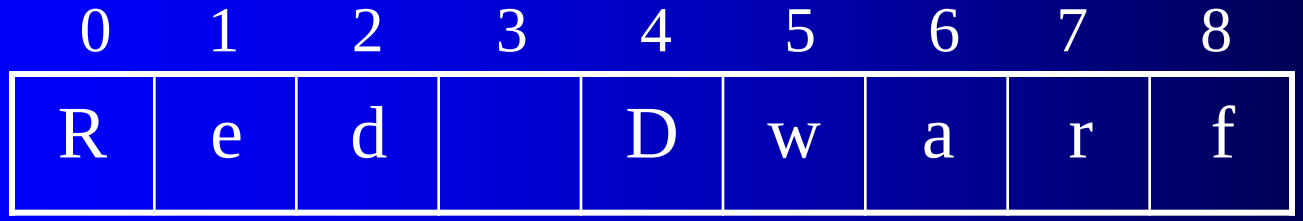
```
"Resist or serve".length( )

motto.length( )

"\u00E9\u00ED\'aan\u00ED\u00EDg\u00D3\u00D3".length( )
```

String indexes

- Strings are stored as a sequence of characters
- Each character is at a particular position
- We can look up the position of a given character or look up the character stored at a specific position (index)
- Index numbering starts with 0



Methods that take parameters

- Many methods require extra data on which to work
- The information a method expects for it to work is called its parameters
- The actual values we give the parameters in a specific call are referred to as its arguments

Methods that take parameters

- The charAt(int) method requires an argument of the integer type with which to work
- It returns the character the index specified by the argument as its result

```
phrase = "Believe to Understand";
char ch = phrase.charAt(5);
System.out.println(ch);
```

- String comparison

Comparing String objects

- equals(String)
  - Returns true if the values of the Strings are exactly the same
- equalsIgnoresCase(String)
  - Same as above, but allows the case of alphabetic characters to be different

Comparing String objects

```
boolean b = "Hello".equals("Hello");
boolean c = "Hello".equals("Hello there");
boolean d = "Hello".equals("hello");

b = "Hello".equalsIgnoreCase("Hello");
c = "Hello".equalsIgnoreCase("Hello there");
d = "Hello".equalsIgnoreCase("hello");
```

Comparing String objects

- compareTo(String)
  - Lexicographic compare
  - Returns an integer representing the relative lexicographic order of the String object and the argument
  - Positive value (if the string object is greater)
  - Zero (if the string object and the argument are the same)
  - Negative value (if the string object is less)

Example

```
String name1 = "fred";
String name2 = "barney";
String name3 = "frederica";
String name4 = "Fred";

int n = name1.compareTo(name2);
int o = name1.compareTo(name3);
int p = name3.compareTo(name4);

System.out.println(n + " " + o + " " + p);
```

- Common string operations

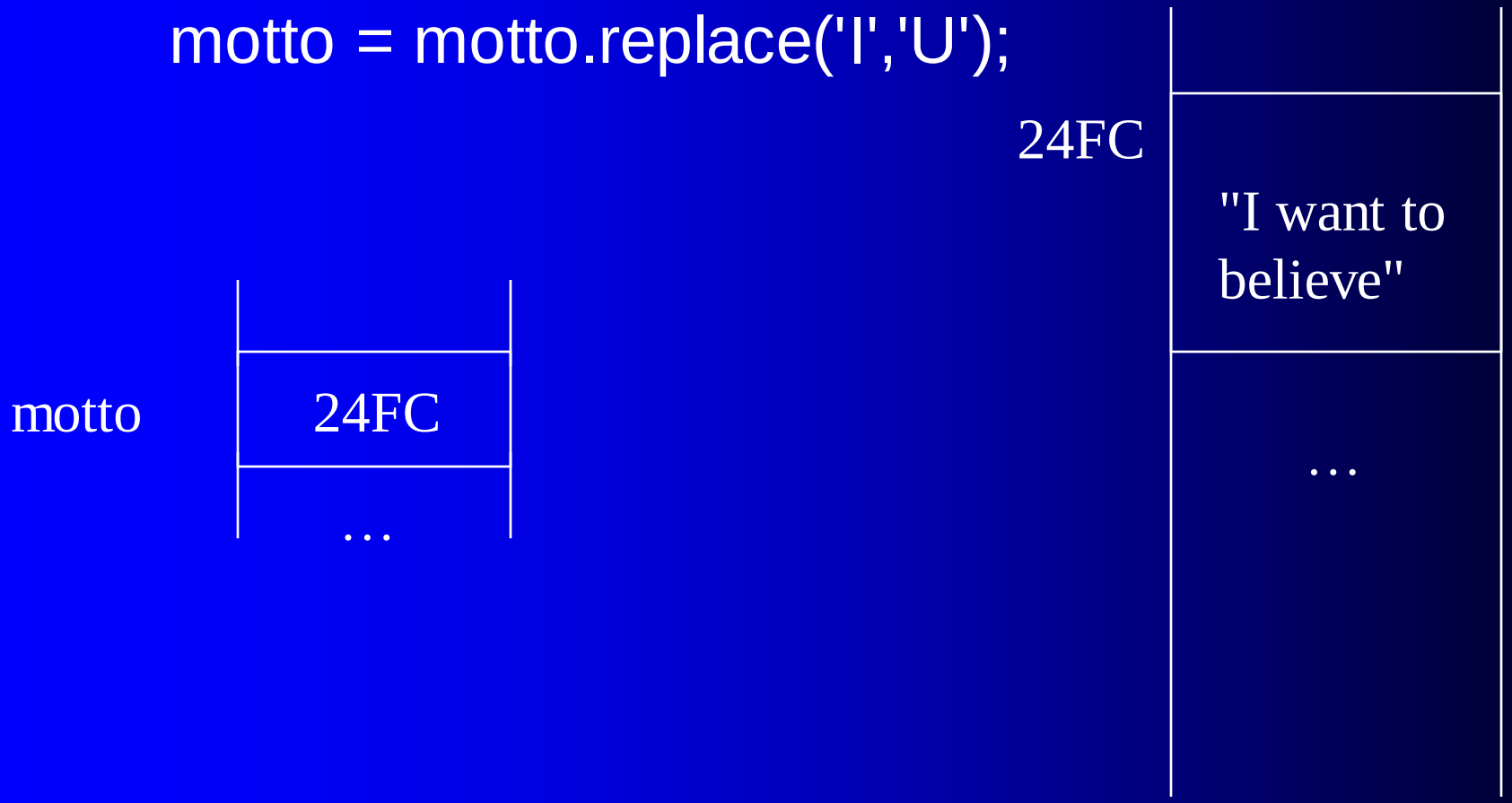
String operations (to replace characters)

- replace(char,char)
  - Creates a new string that is the same as the String object but with all occurrences of the first character, replaced by the second character argument

```
motto = "I want to believe";
motto = motto.replace('I','U');
```

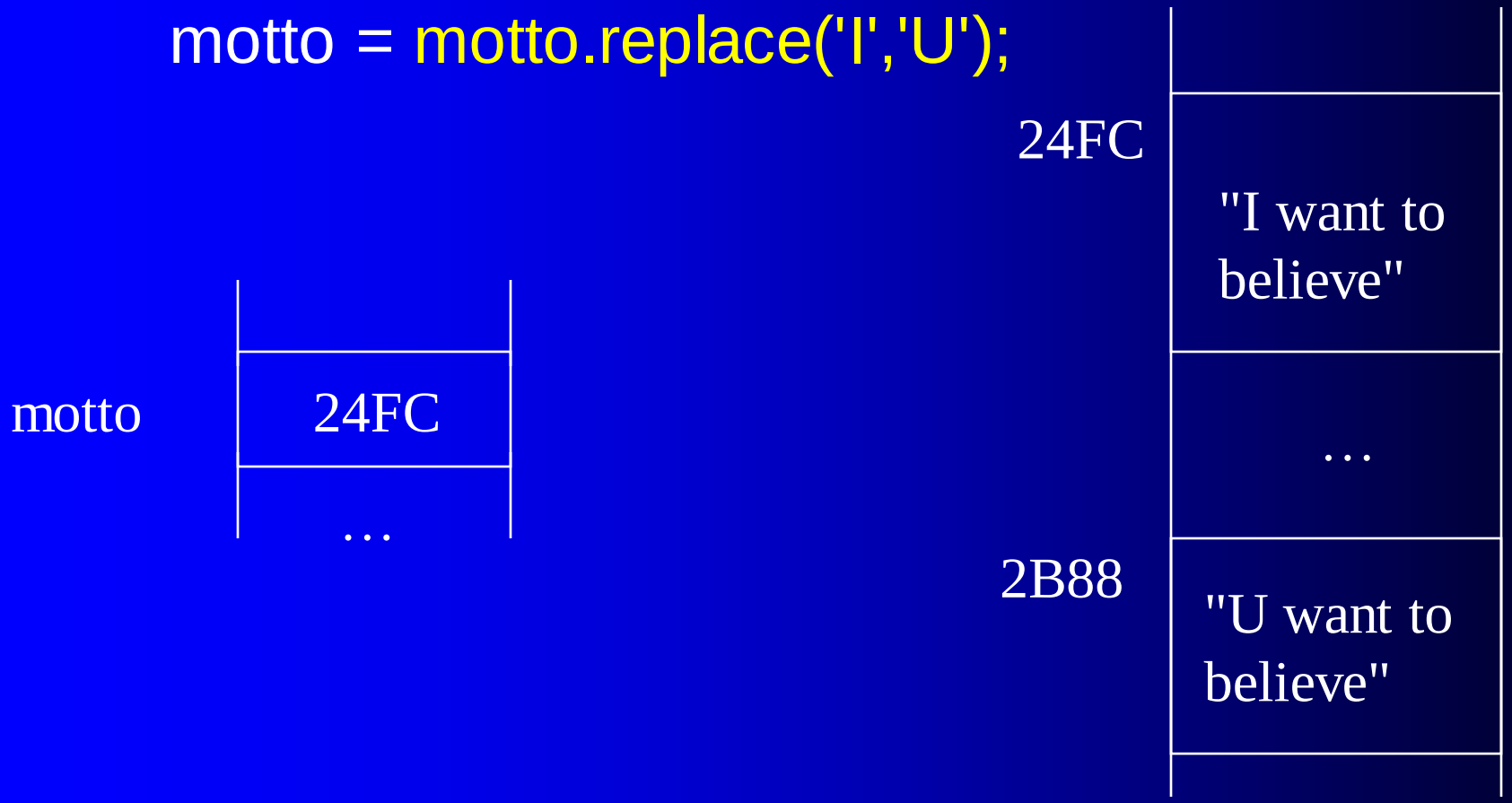
String operations

```
motto = "I want to believe";
motto = motto.replace('I','U');
```



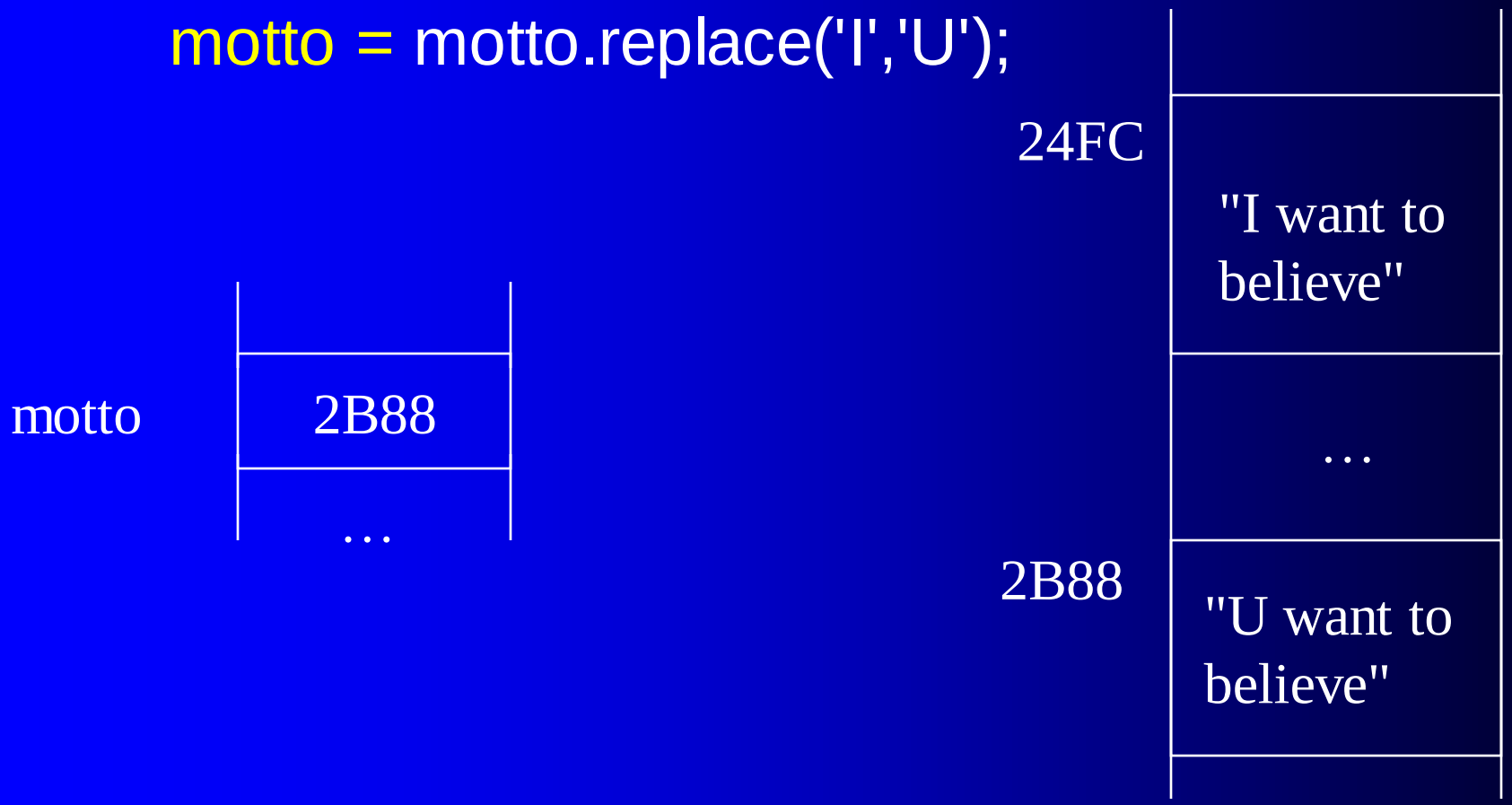
String operations

```
motto = "I want to believe";
motto = motto.replace('I','U');
```



String operations

```
motto = "I want to believe";
motto = motto.replace('I','U');
```



Example

- Replace each occurrence of ‘I’ by ‘U’

```
public class ReplaceDemo
{
    public static void main(String[ ] args)
    {
        String s = "I think therefore I exist!";
        s = s.replace('I', 'U');
        System.out.println(s);
    }
}
```

String operations (to change cases)

- toUpperCase( )
  - Creates a new string like the String object except any alphabetic character is uppercase in the new string

    motto = motto.toUpperCase( );

- toLowerCase( )
  - Creates a new string like the String object except any alphabetic character is lowercase in the new string

    motto = motto.toLowerCase( );

String operations (to trim blanks)

- trim( )
  - Creates a new string like the String object except any whitespace at the start or the end of the String object is removed

```
String motto = "   I want to believe   ";
motto = motto.trim( );
```

String indexes (to search for substring)

- indexOf(String)
  - Takes as an argument a String, and returns the index of the first time the String argument appears in the String object (returns -1 if there is no such substring)

```
String groceries = "Apples, oranges and bananas";
groceries.indexOf("oranges");
```

- lastIndexOf(String)
  - Finds the last occurrence of the String argument

Example

```
public class SubstringDemo
{
    public static void main(String [] args)
    {
        String s = "012ABCxyzABC";

        System.out.println(s.indexOf("ABC"));        // displays 3
        System.out.println(s.lastIndexOf("ABC"));    // displays 9

        System.out.println(s.indexOf("Hello"));      // displays -1
        System.out.println(s.lastIndexOf("Hello"));  // displays -1

        // The indexOf method returns a value that allows us
        // determine if a string contains a particular substring
    }
}
```

String operations (to extract substring)

- substring(int)
  - Creates a new string which contains part of the String object, in this case from the index specified by the argument to the end of the string

```
String m = "I want to believe";
String n = m.substring(3);
```

String operations

- substring(int,int)
  - Creates a new string which contains part of the String object, in this case from the index specified by the first argument to one before the index specified by the second argument

```
String m = "I want to believe";
String n = m.substring(3, 6);
String s1 = “0123456789”;
String s2 = s1.substring(3,6)
```

Next Lecture

- Keyboard input
- Screen output
- The Scanner class
- Documentation and style