

Designing Data

Introduction to Programming and
Computational Problem Solving:
Accelerated Pace

CSE 11

Lecture 3

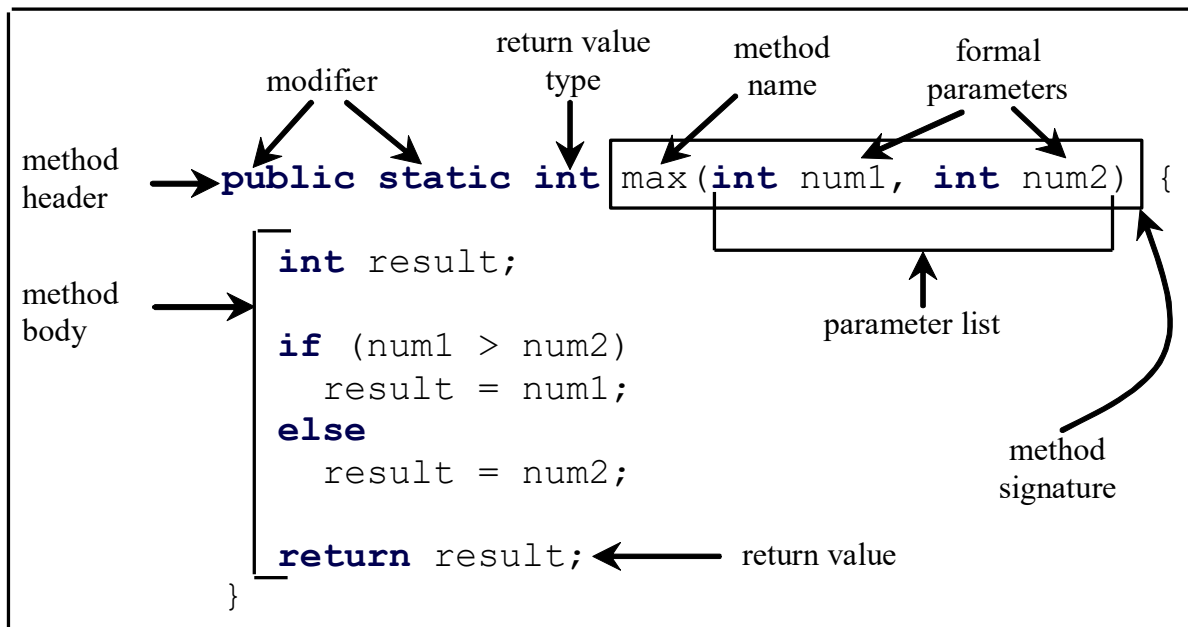
Naming conventions

- Variables and methods: Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name
 - For example, the variables `radius` and `isLeapYear`, and the methods `kick` and `brake`.
- Classes: Capitalize the first letter of each word in the name class (e.g., `WonderWall`)
 - Class names are nouns

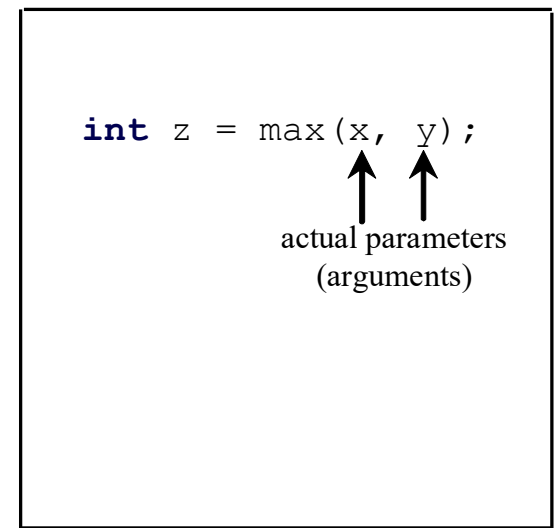
Defining methods

- A method is a collection of statements that are grouped together to perform an operation

Define a method



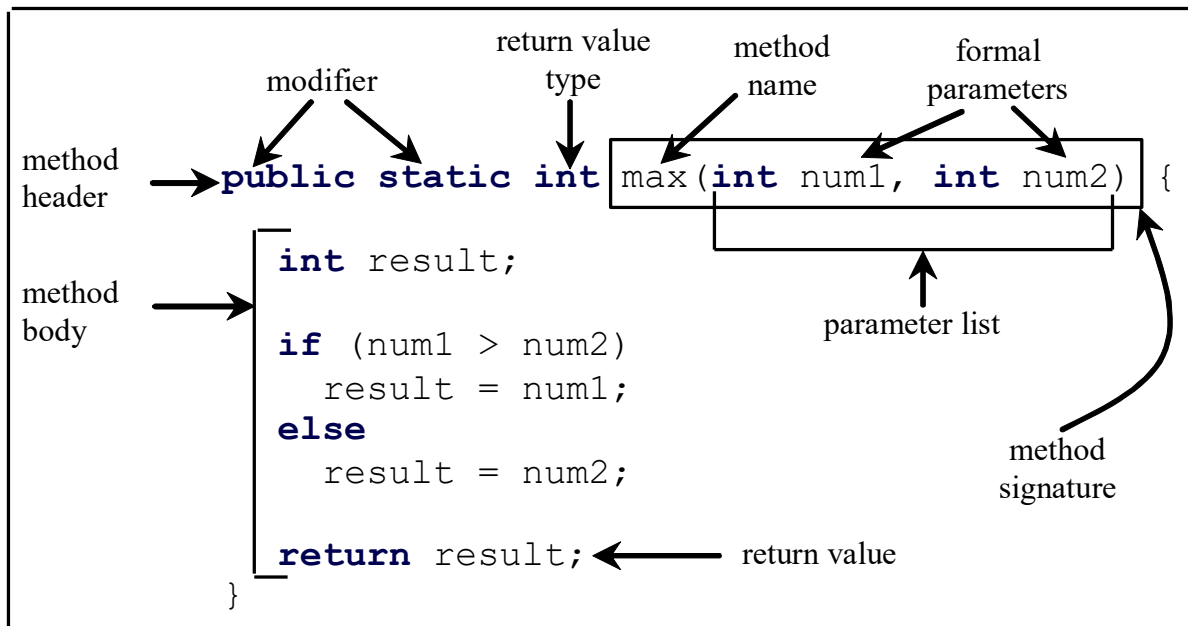
Invoke a method



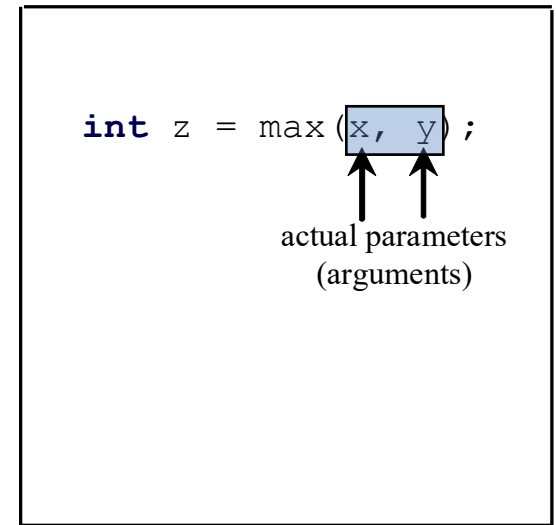
Pass by value

- Java uses **pass by value** to pass arguments to a method
- For example, modifying num1 does not modify x

Define a method



Invoke a method



Scope of local variables

- A local variable is a variable defined inside a method
- Scope is the part of the program where the variable can be referenced
- The scope of a local variable **starts from its declaration** and **continues to the end of the block** that contains the variable
- A local variable must be declared before it can be used
- You can declare a local variable with the same name multiple times in different *non-nesting* blocks in a method, but you cannot declare a local variable twice in nested blocks

Classes

- A Java class uses **variables to define data fields** and **methods to define behaviors**
- Additionally, a class provides a special type of methods, known as **constructors**, which are invoked to construct objects from the class

Records

- Java records act as concise, shallowly immutable data carriers that eliminate boilerplate code for simple classes
 - Automatic members: the compiler automatically generates private final fields, public accessor methods, a canonical constructor (i.e., a constructor that accepts all components of a record in the exact order and type they are declared), and standard implementations of `equals()`, `hashCode()`, and `toString()` (we'll cover these later in the quarter)
<https://docs.oracle.com/en/java/javase/25/language/records.html>
 - For example,

```
record Rect(int width, int height, String color) { }
```

Methods in a record class

- You can declare methods in a record class
- For example

```
record Rect(int width, int height, String color) {  
    boolean hasGreaterWidth(int compareWidth) {  
        return width > compareWidth;  
    }  
}
```

Objects

- An object has a unique identity, state, and behaviors
 - An object is a **unique instance of a class**
 - The **state** of an object consists of a **set of data fields** (also known as properties) with their current values
 - The **behavior** of an object is defined by a **set of methods**

Accessing an object's members

- Use the **object member access operator**
 - Also called the **dot operator** (.)
- Reference the object's data using `objectRefVar.variableName`
- Invoke the object's method using `objectRefVar.methodName(arguments)`

Member variables and methods **do not use the dot operator** to access other member variables and methods **within the current object** (but, when method formal parameters have the same name as a member, then member variables and methods must be accessed a special way; covered later in the quarter).

Instance data fields and methods vs static data fields and methods

- **Instance** data fields and methods **can only be accessed using an object** (i.e., an instance of a class)
 - The syntax to access an **instance data field** is
`objectReferenceVariable.variableName`
 - The syntax to invoke an **instance method** is
`objectReferenceVariable.methodName(arguments)`
- **Static** data fields and methods (i.e., non-instance data fields and methods) can be accessed **without using an object** (i.e., they are not tied to a specific instance of a class)
 - The syntax to access a **static data field** is
`ClassName.variableName`
 - The syntax to invoke a **static method** is
`ClassName.methodName(arguments)`

String type

- The char type only represents one character
- To represent a string of characters, use the String type
- String is a predefined class in the Java library (just like the System class and Scanner class)

<https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/lang/String.html>

- Near all of the methods defined in the String class are instance methods

Simple String methods

Method	Description
<code>length()</code>	Returns the number of characters in this string.
<code>charAt(index)</code>	Returns the character at the specified index from this string.
<code>concat(s1)</code>	Returns a new string that concatenates this string with string <code>s1</code> .
<code>toUpperCase()</code>	Returns a new string with all letters in uppercase.
<code>toLowerCase()</code>	Returns a new string with all letters in lowercase.
<code>trim()</code>	Returns a new string with whitespace characters trimmed on both sides.

- These methods can only be invoked from a specific string instance
 - These methods are called instance methods

String concatenation

```
String s3 = s1.concat(s2); // These two are  
String s3 = s1 + s2;      // equivalent
```

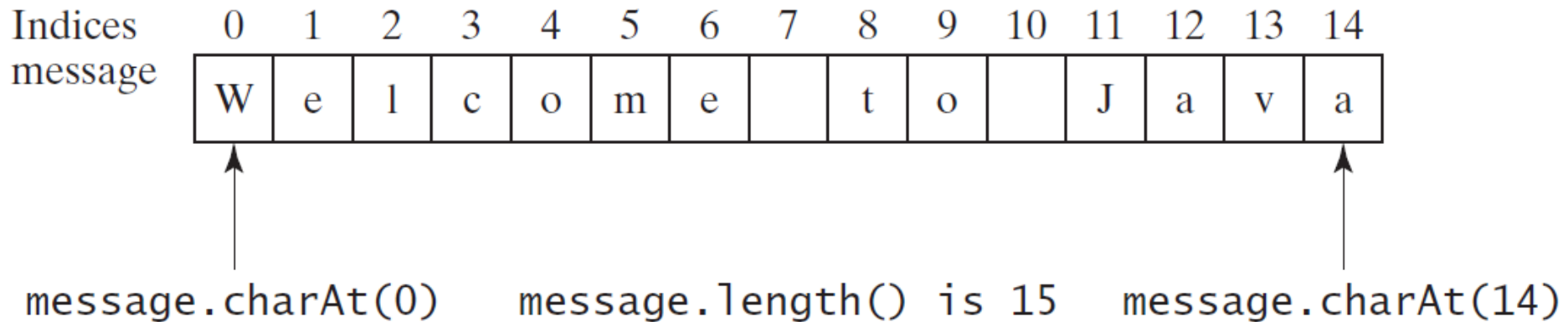
```
// Three strings are concatenated  
String message = "Welcome " + "to " + "Java";
```

```
// String Chapter is concatenated with number 2  
String s = "Chapter" + 2; // s becomes Chapter2
```

```
// String Supplement is concatenated with character B  
String s1 = "Supplement" + 'B'; // s1 becomes SupplementB
```

Getting characters from a string

```
String message = "Welcome to Java";  
System.out.println("The first character in message is "  
    + message.charAt(0));
```

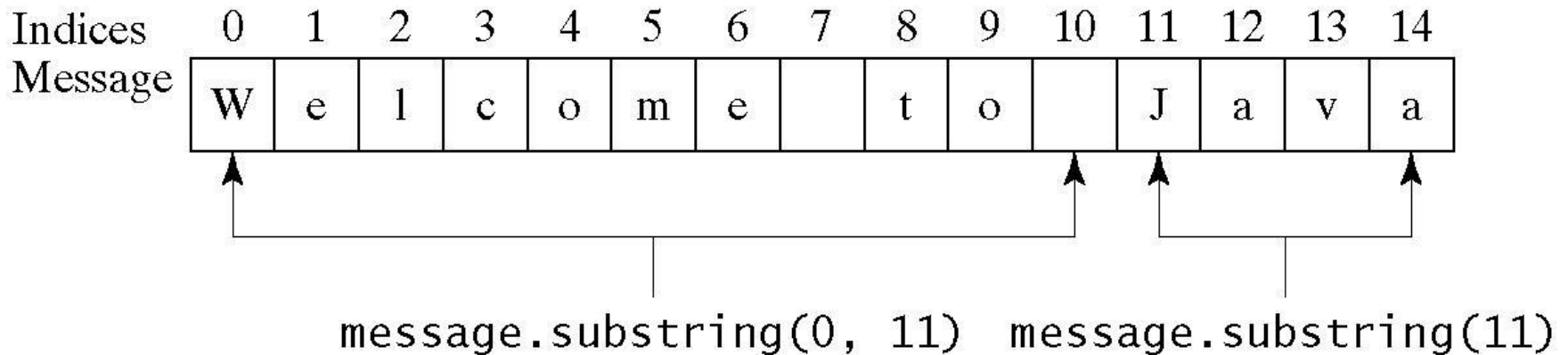


Comparing strings

Method	Description
<code>equals(s1)</code>	Returns true if this string is equal to string <code>s1</code> .
<code>equalsIgnoreCase(s1)</code>	Returns true if this string is equal to string <code>s1</code> ; it is case insensitive.
<code>compareTo(s1)</code>	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than, equal to, or less than <code>s1</code> .
<code>compareToIgnoreCase(s1)</code>	Same as <code>compareTo</code> except that the comparison is case insensitive.
<code>startsWith(prefix)</code>	Returns true if this string starts with the specified prefix.
<code>endsWith(suffix)</code>	Returns true if this string ends with the specified suffix.

Substrings

Method	Description
<code>substring(beginIndex)</code>	Returns this string's substring that begins with the character at the specified <code>beginIndex</code> and extends to the end of the string.
<code>substring(beginIndex, endIndex)</code>	Returns this string's substring that begins at the specified <code>beginIndex</code> and extends to the character at index <code>endIndex - 1</code> . Note that the character at <code>endIndex</code> is not part of the substring.

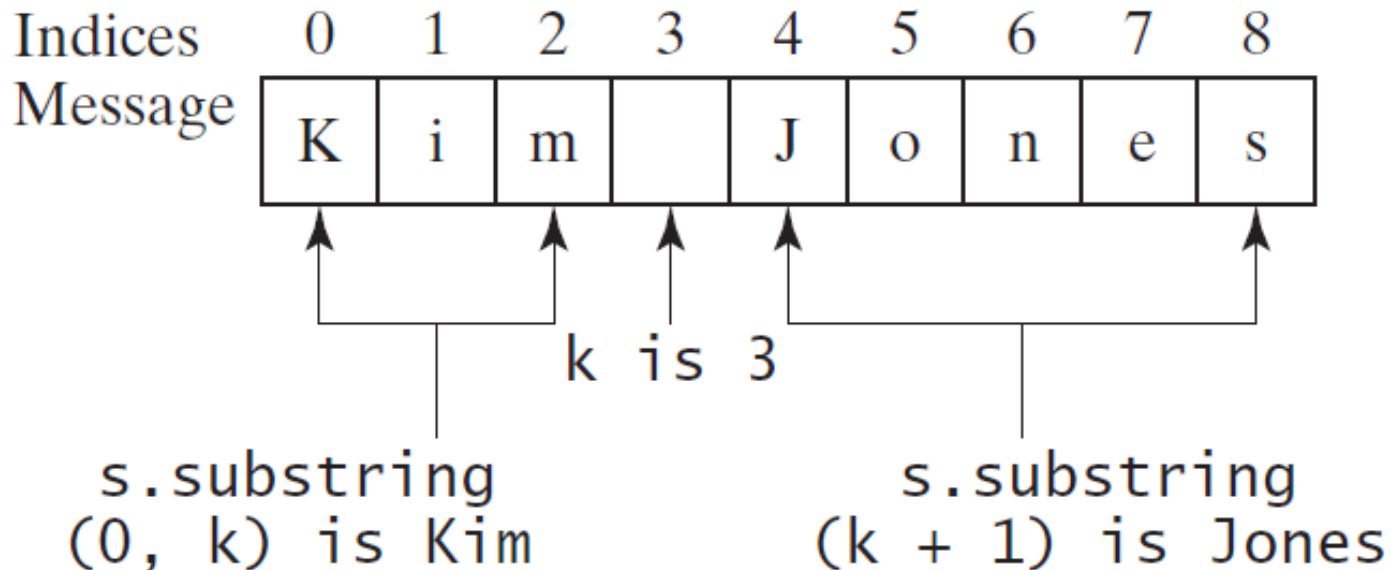


Finding a character or a substring in a string

Method	Description
<code>indexOf(ch)</code>	Returns the index of the first occurrence of <code>ch</code> in the string. Returns <code>-1</code> if not matched.
<code>indexOf(ch, fromIndex)</code>	Returns the index of the first occurrence of <code>ch</code> after <code>fromIndex</code> in the string. Returns <code>-1</code> if not matched.
<code>indexOf(s)</code>	Returns the index of the first occurrence of string <code>s</code> in this string. Returns <code>-1</code> if not matched.
<code>indexOf(s, fromIndex)</code>	Returns the index of the first occurrence of string <code>s</code> in this string after <code>fromIndex</code> . Returns <code>-1</code> if not matched.
<code>lastIndexOf(ch)</code>	Returns the index of the last occurrence of <code>ch</code> in the string. Returns <code>-1</code> if not matched.
<code>lastIndexOf(ch, fromIndex)</code>	Returns the index of the last occurrence of <code>ch</code> before <code>fromIndex</code> in this string. Returns <code>-1</code> if not matched.
<code>lastIndexOf(s)</code>	Returns the index of the last occurrence of string <code>s</code> . Returns <code>-1</code> if not matched.
<code>lastIndexOf(s, fromIndex)</code>	Returns the index of the last occurrence of string <code>s</code> before <code>fromIndex</code> . Returns <code>-1</code> if not matched.

Finding a character or a substring in a string

```
int k = s.indexOf(' ');  
String firstName = s.substring(0, k);  
String lastName = s.substring(k + 1);
```



Next lecture

- `java` and `main`