Using Objects from Existing Classes

Computer Science S-111
Harvard University
David G. Sullivan, Ph.D.

Combining Data and Operations

- The data types that we’ve seen thus far are referred to as *primitive* data types.
  - `int`, `double`, `char`
  - several others

- Java allows us to use another kind of data known as an *object*.

- An object groups together:
  - one or more data values (the object's *fields*)
  - a set of operations (the object's *methods*)

- Objects in a program are often used to model real-world objects.
Combining Data and Operations (cont.)

- Example: an Address object
  - possible fields: street, city, state, zip
  - possible operations: get the city, change the city, check if two addresses are equal

- Here are two ways to visualize an Address object:

| street    | "111 Cummington St." |
| city      | "Boston"              |
| state     | "MA"                  |
| zip       | "02215"               |

getCity()  
changeCity()  ...

Classes as Blueprints

- We've been using classes as containers for our programs.

- A class can also serve as a blueprint – as the definition of a new type of object.

- The objects of a given class are built according to its blueprint.

- Another analogy:
  - class = cookie cutter
  - objects = cookies

- The objects of a class are also referred to as instances of the class.
Class vs. Object

- The Address class is a blueprint:

```java
public class Address {
    // definitions of the fields
    ...
    // definitions of the methods
    ...
}
```

- Address objects are built according to that blueprint:

```
street    "111 Cummington St."
city      "Boston"
state     "MA"
zip       "02215"

street    "240 West 44th Street"
city      "New York"
state     "NY"
zip       "10036"

street    "1600 Pennsylvania Ave."
city      "Washington"
state     "DC"
zip       "20500"
```

Using Objects from Existing Classes

- Later in the course, you'll learn how to create your own classes that act as blueprints for objects.

- For now, we'll focus on learning how to use objects from existing classes.
String Objects

• In Java, a string (like "Hello, world!") is actually represented using an object.
  • data values: the characters in the string
  • operations: get the length of the string, get a substring, etc.

• The String class defines this type of object:

```java
public class String {
    // definitions of the fields
    ...
    // definitions of the methods
    ...
}
```

• Individual String objects are instances of the String class:

| Perry | Hello | object |

Variables for Objects

• When we use a variable to represent an object, the type of the variable is the name of the object's class.

• Here's a declaration of a variable for a String object:

```java
String name;
```

  • we capitalize string, because it's a class name
Creating String Objects

- One way to create a String object is to specify a string literal:
  ```java
  String name = "Perry Sullivan";
  ```

- We create a new String from existing Strings when we use the + operator to perform concatenation:
  ```java
  String firstName = "Perry";
  String lastName = "Sullivan";
  String fullName = firstName + " " + lastName;
  ```

- Recall that we can concatenate a String with other types of values:
  ```java
  String msg = "Perry is " + 6;
  // msg now represents "Perry is 6"
  ```

Using an Object's Methods

- An object's methods are different from the static methods that we've seen thus far.
  - they're called non-static or instance methods

- An object's methods belong to the object. They specify the operations that the object can perform.

- To use a non-static method, we have to specify the object to which the method belongs.
  - use dot notation, preceding the method name with the object's variable:
    ```java
    String firstName = "Perry";
    int len = firstName.length();
    ```

- Using an object's method is like sending a message to the object, asking it to perform that operation.
The API of a Class

- The methods defined within a class are known as the API of that class.
  - API = application programming interface
- We can consult the API of an existing class to determine which operations are supported.
- The API of all classes that come with Java is available here: https://docs.oracle.com/javase/8/docs/api/
  - there’s a link on the resources page of the course website

Consulting the Java API

select the package name (optional)

String is in java.lang
Consulting the Java API

- Scroll down to see a summary of the available methods:

  - `length()`
  - `matches(String regex)`
  - `startsWith(String prefix)`
  - `endsWith(String suffix)`
  - `compareTo(String other)`
  - `indexOf(int ch)`
  - `indexOf(String substr)`
  - `lastIndexOf(String substr)`
  - `lastIndexOf(int ch)`
  - `compareToIgnoreCase(String other)`
  - `toUpperCase()`
  - `toLowerCase()`
  - `trim()`
  - `replace(String old, String new)`
  - `replaceAll(String regex, String replacement)`
  - `format(String format)`
  - `format(float)`
  - `format(double)`
  - `format(long)`
  - `format(int)`
  - `format(boolean)`
  - `format(char)`
  - `format(BigInteger)`
  - `format(BigDecimal)`
  - `format(Number)`

select the class name
Consulting the Java API (cont.)

• Clicking on a method name gives you more information:

```
public int length()
```

**method header**

- Returns the length of this string. The length is equal to the number of Unicode code units in the string.
- Specified by: `length in interface CharSequence`
- Returns:
  - the length of the sequence of characters represented by this object.

**behavior**

• From the header, we can determine:
  • the return type: `int`
  • the parameters we need to supply:
    - the empty `()` indicates that `length` has no parameters

---

Numbering the Characters in a String

• The characters are numbered from left to right, starting from 0.

```
0 1 2 3 4
Perry
```

• The position of a character in a string is known as its **index**.
  • 'P' has an index of 0 in "Perry"
  • 'y' has an index of 4
substring Method

```java
public String substring(int beginIndex, int endIndex)
```

Returns a new string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at `index endIndex - 1`. Thus the length of the substring is `endIndex - beginIndex`.

**String `substring(int beginIndex, int endIndex)`**

- **return type:** 
- **parameters:** 
- **behavior:** returns the substring that:
  - begins at `beginIndex`
  - ends at `endIndex - 1`

**substring Method (cont.)**

- To extract a substring of length $N$, you can just figure out `beginIndex` and do:
  ```java
  substring(beginIndex, beginIndex + $N$)
  ```
- **example:** consider again this string:
  ```java
  String name = "Perry Sullivan";
  ```
  To extract a substring containing the first 5 characters, we can do this:
  ```java
  String first = name.substring(0, 5);
  ```
Review: Calling a Method

• Consider this code fragment:

```java
String name = "Perry Sullivan";
int start = 6;
String last = name.substring(start, start + 8);
```

• Steps for executing the method call:
  1. the actual parameters are evaluated to give:
     ```java
     String last = name.substring(6, 14);
     ```
  2. a frame is created for the method, and the actual parameters are assigned to the formal parameters
  3. flow of control jumps to the method, which creates and returns the substring "Sullivan"
  4. flow of control jumps back, and the returned value replaces the method call:
     ```java
     String last = "Sullivan";
     ```

How should we fill in the blank?

```java
String s = "Strings have methods inside them!";
int len = s.length();
_______________ // get the last character in s
```
charAt Method

- The `charAt()` method that we use for indexing returns a `char`, not a `String`.

- We have to be careful when we use its return value!
  - example: what does this print?
    ```java
    String name = "Perry Sullivan";
    System.out.println(name.charAt(0) + name.charAt(6));
    ```

charAt Method

- Here’s how we can fix this:
  ```java
  String name = "Perry Sullivan";
  System.out.println(name.charAt(0) + name.charAt(6));
  System.out.println('P' + 'S');
  System.out.println("PS");
  ```
Another String Method

String toUpperCase()
returns a new String in which all of the letters in the original String are converted to upper-case letters

• Example:
  String warning = "Start the problem set ASAP!";
  System.out.println(warning.toUpperCase());
  System.out.println("START THE PROBLEM SET ASAP!");

• toUpperCase() creates and returns a new String. It does not change the original String.

• In fact, it's never possible to change an existing String object.

• We say that Strings are immutable objects.

indexOf Method

int indexOf(char ch)
  • return type: int
  • parameter list: (char ch)
  • returns:
    • the index of the first occurrence of ch in the string
    • -1 if the ch does not appear in the string
  • examples:
    String name = "Perry Sullivan";
    System.out.println(name.indexOf('r'));
    System.out.println(name.indexOf('X'));
The Signature of a Method

• The signature of a method consists of:
  • its name
  • the number and types of its parameters

```java
public String substring(int beginIndex, int endIndex)
```

• A class cannot include two methods with the same signature.

Two Methods with the Same Name

• There are actually two String methods named substring:
  ```java
  String substring(int beginIndex, int endIndex)
  
  String substring(int beginIndex)
  ```

  • returns the substring that begins at beginIndex and continues to the end of the string

• Do these two methods have the same signature?

• Giving two methods the same name is known as method overloading.

• When you call an overloaded method, the compiler uses the number and types of the actual parameters to figure out which version to use.
Console Input Using a Scanner Object

• We’ve been printing text in the console window.

• You can also ask the user to enter a value in that window.
  • known as console input

• To do so, we use a type of object known as a Scanner.
  • recall PS 2

Packages

• Java groups related classes into \textit{packages}.

• Many classes are part of the \texttt{java.lang} package.
  • examples: \texttt{String, Math}
  • We don’t need to tell the compiler where to find these classes.

• If a class is in another package, we need to use an \texttt{import} statement so that the compiler will be able to find it.
  • put it \textit{before} the definition of the class

• The \texttt{Scanner} class is in the \texttt{java.util} package, so we do this:

\begin{verbatim}
import java.util.*;
public class MyProgram {
  ...
\end{verbatim}
Creating an Object

- String objects are different from other objects, because we're able to create them using literals.

- To create an object, we typically use a special method known as a constructor.

- Syntax:
  
  `<variable> = new <ClassName>(<parameters>);`
  
  or
  
  `<type> <variable> = new <ClassName>(<parameters>);`

- To create a scanner object for console input:
  
  ```java
  Scanner console = new Scanner(System.in);
  ```
  
  the parameter tells the constructor that we want the scanner to read from the standard input (i.e., the keyboard)

Scanner Methods: A Partial List

- String `next()`
  - read in a single "word" and return it

- `nextInt()`
  - read in an integer and return it

- `nextDouble()`
  - read in a floating-point value and return it

- String `nextLine()`
  - read in a "line" of input (could be multiple words) and return it
Example of Using a Scanner Object

- To read an integer from the user:
  
  ```java
  Scanner console = new Scanner(System.in);
  int numGrades = console.nextInt();
  ```

- The second line causes the program to pause until the user types in an integer followed by the [ENTER] key.

- If the user only hits [ENTER], it will continue to pause.

- If the user enters an integer, it is returned and assigned to numGrades.

- If the user enters a non-integer, an exception is thrown and the program crashes.

Example Program: GradeCalculator

```java
import java.util.*;
public class GradeCalculator {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("Points earned: ");
        int points = console.nextInt();
        System.out.print("Possible points: ");
        int possiblePoints = console.nextInt();

        double grade = points/(double)possiblePoints;
        grade = grade * 100.0;

        System.out.println("grade is "+ grade);
    }
}
```
Important Note About Console Input

• When writing an interactive program that involves user input in methods other than main, you should:
  • create a single Scanner object in the first line of the main method
  • pass that object into any other method that needs it

• This allows you to avoid creating multiple objects that all do the same thing.

• It also facilitates our grading, because it allows us to provide a series of inputs using a file instead of the keyboard.

Important Note About Console Input (cont.)

• Example:
  public class MyProgram {
    public static void main(String[] args) {
      Scanner console = new Scanner(System.in);
      String str1 = getString(console);
      String str2 = getString(console);
      System.out.println(str1 + " " + str2);
    }
    
    public static String getString(Scanner console){
      System.out.print("Enter a string: ");
      String str = console.next();
      return str;
    }
  }
}
What's Wrong with the Following?

```java
class LengthConverter {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        int cm = getInches(console) * 2.54;
        System.out.println(getInches(console) + " inches = " + cm + " cm");
    }

    public static int getInches(Scanner console) {
        System.out.print("Enter a length in inches: ");
        int inches = console.nextInt();
        return inches;
    }
}
```

Exercise: Analyzing a Name: First Version

```java
class NameAnalyzer {
    public static void main(String[] args) {
        String name = "Perry Sullivan";
        System.out.println("full name = " + name);

        int length = name.length();
        System.out.println("length = " + length);

        String first = name.substring(0, 5);
        System.out.println("first name = " + first);

        String last = name.substring(6);
        System.out.println("last name = " + last);
    }
}
```
Making the Program More General

- Would the code work if we used a different name?

```java
import java.util.*;

public class NameAnalyzer {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        String name = console.nextLine();
        System.out.println("full name = " + name);

        int length = name.length();
        System.out.println("length = " + length);

        String first = name.substring(0, 5);
        System.out.println("first name = " + first);

        String last = name.substring(6);
        System.out.println("last name = " + last);
    }
}
```

Breaking Up a Name

- Given a string of the form "firstName  lastName", how can we get the first and last names, without knowing how long it is?

- Pseudocode for what we need to do:

- What string methods can we use? Consult the API!

- Code:
Static Methods for Breaking Up a Name

• How could we rewrite our name analyzer to use separate methods for extracting the first and last names?

```java
public static _______ firstName(_______________) {
}

public static _______ lastName(_______________) {
}
```

Using the Static Methods

• Given the methods from the previous slide, what would the main method now look like?

```java
public static void main(String[] args) {
    Scanner console = new Scanner(System.in);
    String name = console.nextLine();
    System.out.println("full name = " + name);

    int length = name.length();
    System.out.println("length = " + length);
}
```
Processing a String One Character at a Time

• Write a method for printing the name vertically, one char per line.

```java
import java.util.*;
public class NameAnalyzer {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        String name = console.nextLine();
        System.out.println("full name = " + name);
        ...
        printVertical(name);
    }
    public static _____ printVertical(_______________){
        for (int i = 0; i < _______________; i++) {
        }
    }
}
```

Scanner Objects and Tokens

• Most Scanner methods read one token at a time.

• Tokens are separated by whitespace (spaces, tabs, newlines).
  • example: if the user enters the line
    
    ```text
    wow, I slept for 9 hours!\n    ```

  there are six tokens:
  • wow,
  • I
  • slept
  • for
  • 9
  • hours!

  newline character, which you get when you hit [ENTER]
Scanner Objects and Tokens (cont.)

- Consider the following lines of code:

  ```java
  System.out.print("Enter the length and width: ");
  int length = console.nextInt();
  int width = console.nextInt();
  ```

- Because the `nextInt()` method reads one token at a time, the user can either:
  - enter the two numbers on the same line, separated by one or more whitespace characters
    ```
    Enter the length and width: 30 15
    ```
  - enter the two numbers on different lines
    ```
    Enter the length and width: 30
    15
    ```

nextLine Method

- The `nextLine()` method does not just read a single token.

- Using `nextLine` can lead to unexpected behavior, for reasons that we'll discuss later on.

- Avoid it for now!
Additional Terminology

• To avoid having too many new terms at once, I've limited the terminology introduced in these notes.

• Here are some additional terms related to classes, objects, and methods:
  • *invoking* a method = calling a method
  • method *invocation* = method call
  • the *called object* = the object used to make a method call
  • *instantiate* an object = create an object
  • *members* of a class = the fields and methods of a class