Defining Classes and Methods (Savitch, Chapter 5)

TOPICS

- Java methods
- Java objects
- Static keyword
- Parameter passing
- Constructors

Methods

- A **method** (a.k.a. function, procedure, routine) is a piece of code that performs a useful action
  - You defined a method called 'main'.
  - When you run a Java program, it always begins by running the main method.
- A **method** can also return a value to the program that called them
  - More details in a minute…

Mysteries Revealed

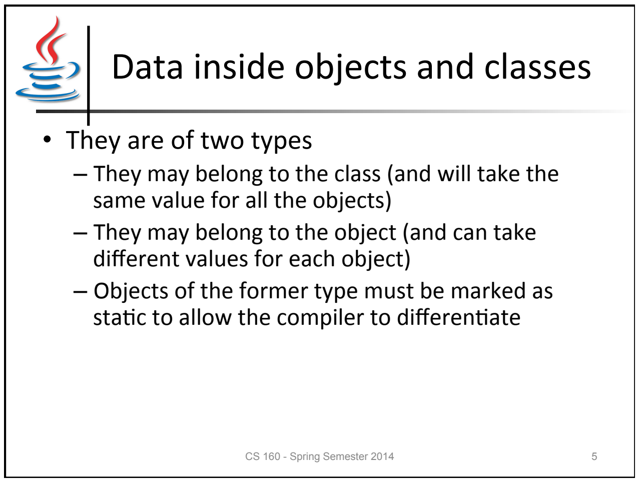
```
public class Temperature {
   public static void main(String[] args) {
      // your code here
   }
}
```

In our recitations and assignments, you define classes (e.g. P1, R1).

You also define a method called 'main' that takes an array of Strings as its arguments

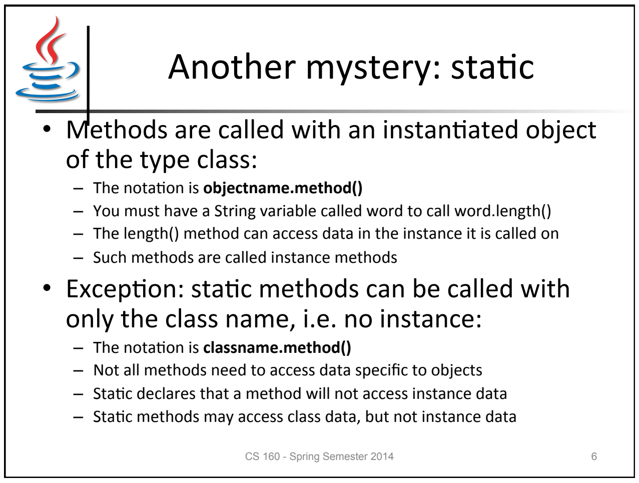
Terminology

- A *class* is a data type
  - Combines variables with methods
- An *object* is an instance of a class
  - Must be explicitly created in program
- Creating an object is called *instantiation*
  - This involves use of **new** operator



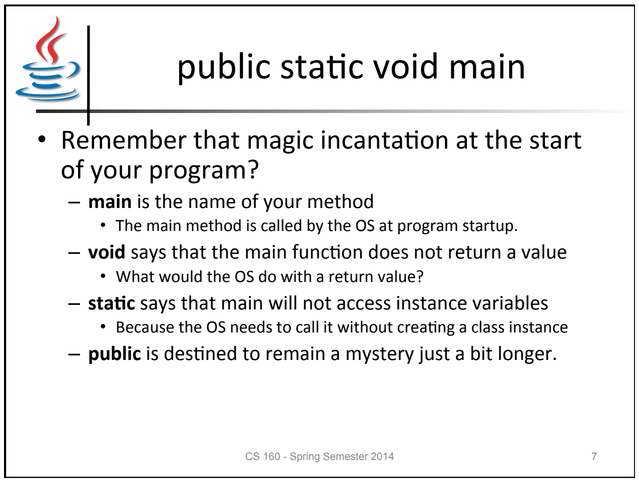
Data inside objects and classes

- They are of two types
  - They may belong to the class (and will take the same value for all the objects)
  - They may belong to the object (and can take different values for each object)
  - Objects of the former type must be marked as static to allow the compiler to differentiate



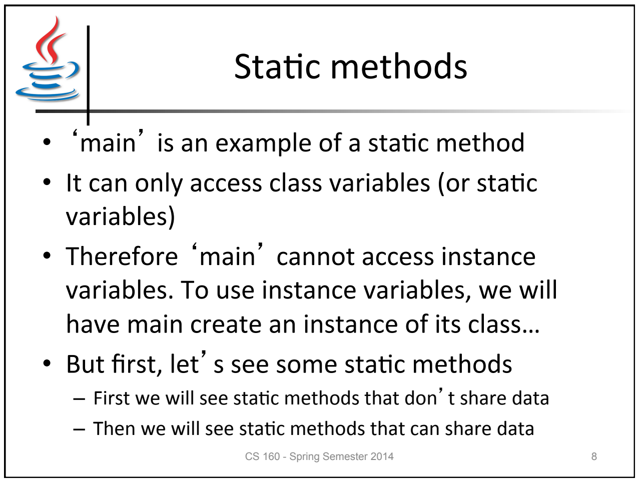
Another mystery: static

- Methods are called with an instantiated object of the type class:
  - The notation is **objectname.method()**
  - You must have a String variable called word to call word.length()
  - The length() method can access data in the instance it is called on
  - Such methods are called instance methods
- Exception: static methods can be called with only the class name, i.e. no instance:
  - The notation is **classname.method()**
  - Not all methods need to access data specific to objects
  - Static declares that a method will not access instance data
  - Static methods may access class data, but not instance data



public static void main

- Remember that magic incantation at the start of your program?
  - **main** is the name of your method
    - The main method is called by the OS at program startup.
  - **void** says that the main function does not return a value
    - What would the OS do with a return value?
  - **static** says that main will not access instance variables
    - Because the OS needs to call it without creating a class instance
  - **public** is destined to remain a mystery just a bit longer.



Static methods

- 'main' is an example of a static method
- It can only access class variables (or static variables)
- Therefore 'main' cannot access instance variables. To use instance variables, we will have main create an instance of its class…
- But first, let's see some static methods
  - First we will see static methods that don't share data
  - Then we will see static methods that can share data

Communication between calling and called methods

- Method parameters:
  - Method declares a parameter "formal parameter" to state what can be provided by the calling program.

public static String reverseCase (String s1)

- Indicates the calling program must specify a String

public static int returnRandom()

- Indicates the calling program specifies no parameters

Communication between calling and called methods

- Method return type and value:
  - Can return void (i.e., nothing)
  - Can return a type (e.g., int, char, String, etc)
    - If a type is returned, there must be a return statement in the method body
    - There must be a return for each reachable part of the code
  - Return type must match in calling program

Communication between calling and called methods

public String reverseCase (String s1)

public int returnRandom()

- Calling method:
  - Supplies arguments that must match the type of the parameters in the method declaration
  - Uses the return value to do something
  - Return value must match type of variable

System.out.print(reverseCase(strname));

int i = returnRandom();

Caution: Pass by value

- What do you expect this to print?

```
public class PassByValue  {
      public static void main(String[] args)  {
            int num = 100;
            increment(num);
            System.out.println("After calling increment, num is " + num);
      }
      public static void increment(int n) {  n++; }
}
```

- The value of the argument is copied. Any changes to the copy are not reflected in the original argument.

Caution: Pass by value

- Another example

```
public class PassByValueString {
    public static void main(String[] args) {
        String word = new String("Good morning");
        changeGreeting(word);
        System.out.println("After calling changeGreeting, word is " + word);
    }

    public static void changeGreeting(String w) {
        w = new String("Good night");
    }
}
```

- Greeting remains unchanged

Incorrect Swapping

```
public class Swapper {
    public static void main(String[] args) {
        String s1 = "Martin";
        String s2 = "Scorcese";
        swap(s1, s2);
        System.out.println("main: After swap, s1=" +s1+ " and s2=" +s2);
    }
    public static void swap(String x, String y) {
        System.out.println("swap: Before swap, x=" +x+ " and y=" +y);
        String temp = x; x = y; y = temp;
        System.out.println("swap: After swap, x=" +x+ " and y=" +y);
    }
}
```

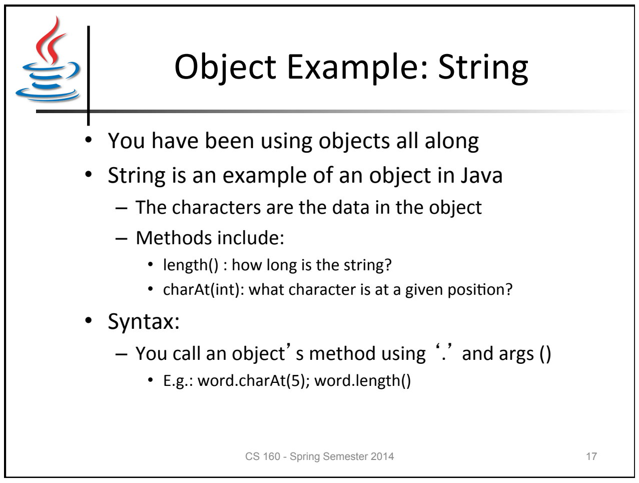
• Nothing gets swapped!

Use methods for subtasks

- The general rule is:
  – Break subtasks into tasks until tasks are trivial
  – Every subtask is a method
  – Some methods (subtasks) may call others

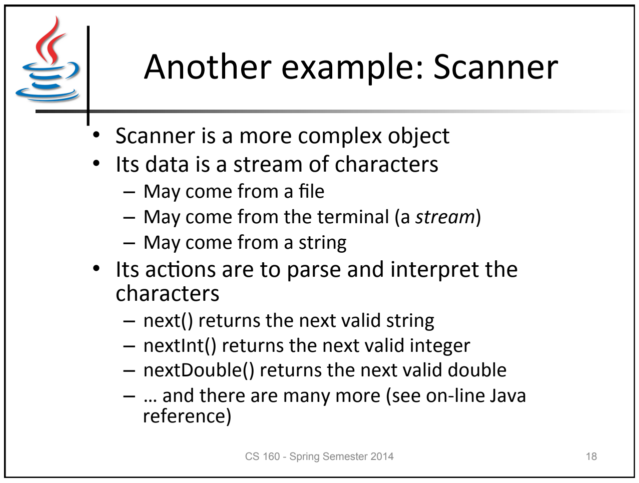
Objects

- An object in Java is
  – A set of *methods* (think: functions)
  – A set of *members* (think: variables)
- Fancy CS buzzwords:
  – Objects *encapsulate data* and *functionality*
  – Objects *encapsulate behavior* and *state*



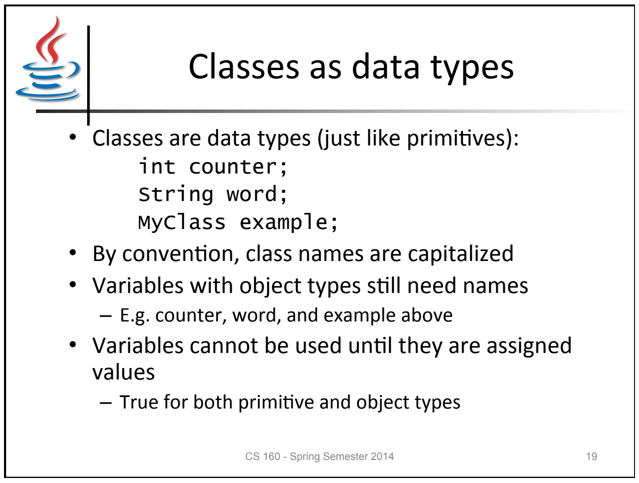
Object Example: String

- You have been using objects all along
- String is an example of an object in Java
  - The characters are the data in the object
  - Methods include:
    - length() : how long is the string?
    - charAt(int): what character is at a given position?
- Syntax:
  - You call an object's method using '.' and args ()
    - E.g.: word.charAt(5); word.length()



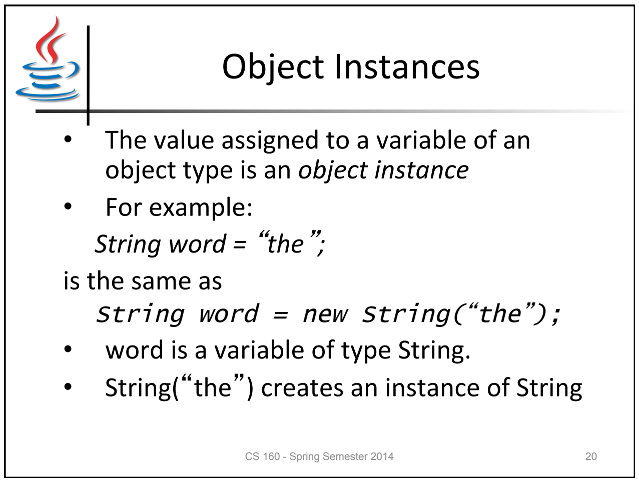
Another example: Scanner

- Scanner is a more complex object
- Its data is a stream of characters
  - May come from a file
  - May come from the terminal (a *stream*)
  - May come from a string
- Its actions are to parse and interpret the characters
  - next() returns the next valid string
  - nextInt() returns the next valid integer
  - nextDouble() returns the next valid double
  - … and there are many more (see on-line Java reference)



Classes as data types

- Classes are data types (just like primitives):
```
int counter;
String word;
MyClass example;
```
- By convention, class names are capitalized
- Variables with object types still need names
  - E.g. counter, word, and example above
- Variables cannot be used until they are assigned values
  - True for both primitive and object types



Object Instances

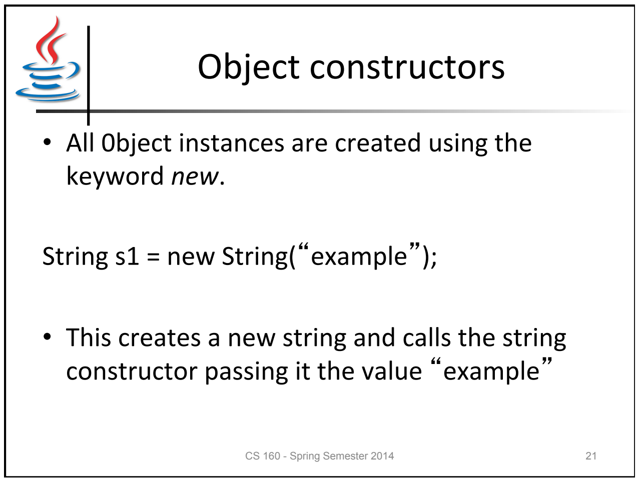
- The value assigned to a variable of an object type is an *object instance*
- For example:

String word = "the";

is the same as

String word = new String("the");

- word is a variable of type String.
- String("the") creates an instance of String



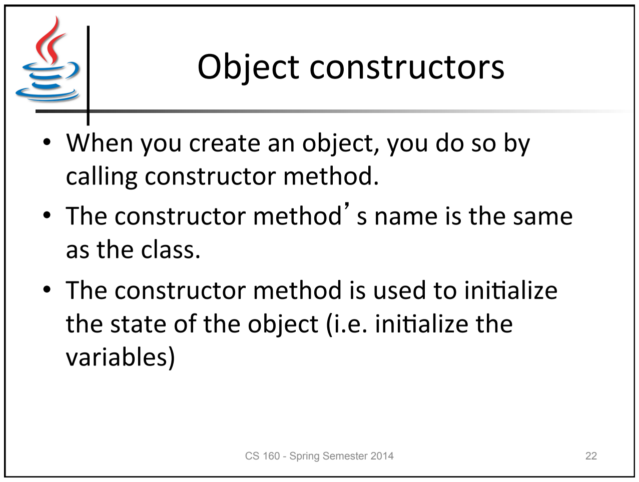
Object constructors

- All Object instances are created using the keyword *new*.

String s1 = new String(“example”);

- This creates a new string and calls the string constructor passing it the value “example”

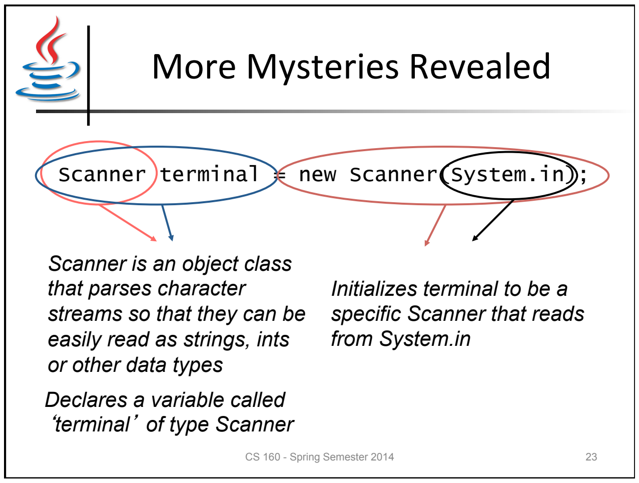
CS 160 - Spring Semester 2014 21



Object constructors

- When you create an object, you do so by calling constructor method.
- The constructor method’s name is the same as the class.
- The constructor method is used to initialize the state of the object (i.e. initialize the variables)

CS 160 - Spring Semester 2014 22



More Mysteries Revealed

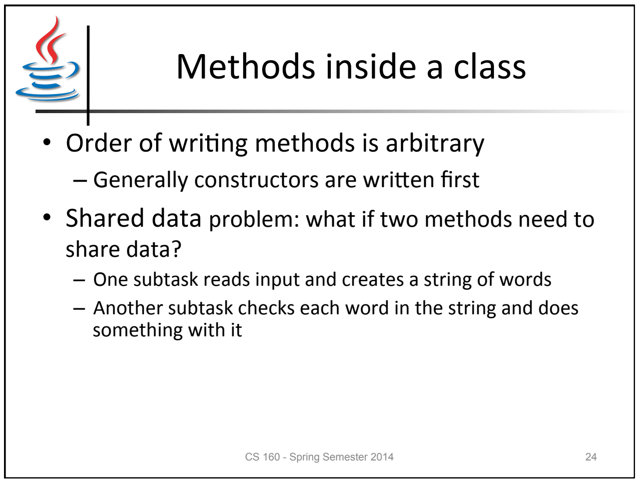
```
Scanner terminal = new Scanner(System.in);
```

Scanner is an object class that parses character streams so that they can be easily read as strings, ints or other data types

Declares a variable called ‘terminal’ of type Scanner

Initializes terminal to be a specific Scanner that reads from System.in

CS 160 - Spring Semester 2014 23



Methods inside a class

- Order of writing methods is arbitrary
  – Generally constructors are written first
- Shared data problem: what if two methods need to share data?
  – One subtask reads input and creates a string of words
  – Another subtask checks each word in the string and does something with it

CS 160 - Spring Semester 2014 24

Solution #1

- Method1 for subtask 1 returns a value,v
- Method2 for subtask 2 uses the value,v
- Example:

```
public static void main(String[] args)  {
     String wordList =   readInput();
     processWords(wordList);
 }
```

Solution #2

- Use instance variables
  - Define String wordList; as an instance variable
  - Any method of a class can access its variables
    - readInput() can create & write the string
    - processWords() can access it

Data Variables in Classes

- How does a method access data in a class?
  - Every method can access the class instance it is called on
    - Think of word.length(); it can access the data in the string 'word'
    - Think of the class instance as a 'hidden' argument to the method
  - Class variables look like any other variables in the code of a method
    - They do not need to be 're-declared'

Simple example

```
public class Course {
    String department, number;

    public Course(String dept, String num) {
        department = dept;
        number = num;
    }
    public String getFullName(){
        return new String(department + " " + number);
    }
    public static void main(String[] args) {
        Course c1 = new Course("CS", "160");
        System.out.println(c1.getFullName());
    }
}
```

Classes

- Classes are the basis of object-oriented (OO) programming.
- They *encapsulate* functionality to form powerful *abstractions* of real world objects.
- What can classes be used for? Classes have many different uses, for example:
  - Data Structures
  - Code Libraries
  - Java Programs
  - Complex Objects

Classes as Data Structures

- Just like a *struct* in C and C++, for example:

```
public class Student {
        public String firstName;
        public String lastName;
        public Date birthDate;
        public Address homeAddress;
        public double gradePointAverage;
}
```

Classes as Code Libraries

- Just like a *library* in a procedural language like C or C++, for example:

```
public class Math {
        public static final double PI = 3.14159;
        public static double sin(double a) {…}
        public static double exp(double a) {…}
        public static double log(double a) {…}
        public static double sqrt(double a) {…}
}
```

Classes as Small Programs

- Just like a *program* in a procedural language like C or C++, for example:

```
public class MySmallProgram{
        public static void main(String args[]) {
                System.out.println("Hello, World!");
        }
}
```

Classes as (Large) Programs

- Just like a *program* in a procedural language like C or C++, for example:

```
public class MyLargeProgram{
        // lots of data
        public static void main(String args[]) {
                // lots of code
        }
        // lots of methods
}
```

Classes as Complex Objects

- No comparable example in a procedural language like C or Pascal!

```
public class MyClass {
        // lots of class variables (static)
        // lots of instance variables (non-static)
        // no main method
        // lots of class methods (static)
        // lots of instance methods (non-static)
}
```

Using Different Class Types

```
// Data Structure
Student students[] = new Student[100];
students[0].firstName = “Christopher”;
// Code Library
System.out.println(Math.sin(1.0));
// Java Programs
$ java MySmallProgram
// Complex Objects
MyClass myClass = new MyClass();
myClass.initialize();
```