Methods

• A **method** (a.k.a. function, procedure, subroutine) is a piece of code that performs a useful action.
  – Up to this point you have (for the most part) put the entire Java program in a single method called **main**.
  – The **main** method is special only in that it is the entry point when you start a Java program.
  – Writing large programs in this manner is impractical for a number of reasons.

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 parameters.
Method Types

- When you use a method you "invoke" or "call" it from another method.
- Two kinds of Java methods
  - Perform some action and return a single item
  - Perform some and return nothing
- The method **main** is a **void** method
  - Invoked by the system, not the application
  - Does not return anything

Calling Methods

- Calling a method that returns an item
  - For **void** method, invoke using object name:
    - `object.method();`
  - When method returns a value:
    - `int value = object.method();`
  - Use the variable value anywhere any other literal or variable value can be used
  - Return values can be of any data type:
    - Primitive types, objects, collections

Defining **void** Methods

- Consider method to print something, does not require any data:
  ```java
  public void printMethod(){
      System.out.println("Student Name: ");
      System.out.println("Student EID: ");
  }
  ```
- Method definitions reside in class definition
  - Can be called only on objects of that class

Method Declarations

- **public** methods can be called from inside or outside the class
- **private** methods can be called only from inside the class
- Data type specifies return type, **void** means no return value
- Heading includes parameters in ( )
- Body enclosed in braces {  }
• Consider method `getMonthsInYear()`

```java
public int getMonthsInYear()
{
    int monthsInYear = 12;
    return monthsInYear;
}
```

• Heading declares type of return value
• Last statement executed is `return`
• Parameter list is empty

### Return Values

### Local Variables

• Variables declared inside a method are called *local* variables
  – May be used only inside the method, i.e. they are valid only in the *scope* of the method
  – All variables declared in method `main` are local to `main`
• Local variables having the same name and declared in different methods are actually different variables!

### Local Example

```java
public double sin(double angle) {
    double value = Math.sin(angle);
    return value;
}
```

```java
public double cos(double angle) {
    double value = Math.cos(angle);
    return value;
}
```

### Blocks

• Recall compound statements
  – Enclosed in braces `{ }`
• When you declare a variable within a compound statement
  – The compound statement is called a *block*
  – The *scope* of the variable is from its declaration to the end of the block
• Variables declared outside the block are usable outside and inside the block
Passing Parameters

• Method declaration must include a list of parameters and their types
  
  ```java
  public double sin(double angle) {
    return Math.sin(angle);
  }
  ```

  • Empty list means no parameters
  • Parameters are separated by commas

Method Parameters

• Note the declaration
  
  ```java
  public double computeInterest(double rate)
  ```

  – The formal parameter or argument is `rate`

• Calling the method
  
  ```java
  double interest = obj.computeInterest(5.9);
  ```

  – The actual parameter is 5.9

Primitive Parameters

• Parameter names are local to the method
• When method invoked
  – Each parameter initialized to value in corresponding actual parameter
  – Primitive actual parameter cannot be altered by invocation of the method
• Automatic type conversion performed
  
  ```
  byte -> short -> int -> long -> float -> double
  ```

Method Examples

• `public double sin(double angle)`
• `public double cos(double angle)`
• `public char charAt(int index)`
• `public int indexOf(char c)`
• `public int minimum(int i, int j)`
• `public String toLower(String s)`
• `public int[] getArray()`
public and private

- **public**
  - Can access the class, method, or data by name outside defining class
- **private**
  - Can access the class, method, or data by name only inside defining class
  - Classes generally specified as **public**
  - Instance variables usually are **private**
  - Methods can be **public** or **private**

Pre-condition Comment

- **Precondition comment**
  - States conditions that must be true before method is invoked
- **Example**
  ```java
double squareRoot(double value) {
    // Parameter value must be > 0
    assert (value >0);
    ...
}
```

Post-condition Comment

- **Postcondition comment**
  - Tells what will be true after method executed
- **Example**
  ```java
double squareRoot(double value) {
    root = ...
    // Return value is the positive square root of the value
    assert(abs(root*root-value)<EPSILON);
}
```

Objects and classes

- A class is a blueprint for objects:
  - String class provides an abstraction
  - Objects of this class can be instantiated
    - e.g. String firstName = new String();
    - firstName is an object (instance) of class String
    - firstName is used to invoke methods in class
- You can define your own classes with
  - Data (e.g., internal private data)
  - Methods
Accessors and Mutators

- When instance variables are private, class must provide methods to access them:
  - Typically named `getSomeValue`
  - Referred to as an accessor (getter) method
- Must also provide methods to change the values of the private instance variable
  - Typically named `setSomeValue`
  - Referred to as a mutator (setter) method

Methods Calling Methods

- A method body may call any other method
- If the method is within the same class
  - Need not use the class prefix of object that is being called
  - `this` keyword is assumed, which is shorthand for the class we’re currently in

Instance Data

- Local data resides in a method, class data resides in the class
- Data defined in the class can be of two types:
  - Data may belong to the class (and will take the same value for all the objects)
  - Data may belong to the object (and can take different values for each instance)

Instance Data: Example

```java
public class P1 {
    public int int0;
    private double real0;
}
```

• `int0` visible outside of class,
• `real0` is not, it is private to the class and can only be accessed from within the class
Instance Methods

- Notation: objectname.method()
- Must be called on an object instantiated (or created) from a class
- Most objects are instantiated with the new keyword: String word = new String(“Whatever”);
- For example, calling word.length() requires an object of type String called word
- The length() method accesses the data for the specific instance it is called on

Communication between calling and called methods

- Method parameters:
  - Methods define a list of formal parameters to state what must be provided by the calling method.
  
```java
public String reverseCase(String s1)
```
  - Indicates the calling program must specify or “pass in” a String

```java
public int returnRandom()
```
  - Empty parentheses indicate the calling program specifies no parameters

Communication between calling and called methods

- Method return type and value:
  - Can return void (e.g. nothing), or can return a type (e.g. int, double, char, String, ...)
    - If a value is returned, there must be a return statement in the method body
    - There must be a return for each possible path through the code or compiler will complain
  - Not limited to returning one primitive, could be a collection or object with lots of data
  - Return type must match assignment in calling method

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 will be explained later
  - public allows access to main outside the class
    - Again the OS needs this access to start the program
Communication between calling and called methods

```java
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 by storing it, printing it, or using it for a calculation, must match type.
System.out.print(reverseCase(str));
int random = returnRandom();
```

Caution: Pass by value

```java
• What do you expect this to print?
public class PassByValue {
    public static void main(String[] args) {
        int number = 100;
        increment(number);
        System.out.println("Number: " + number);
    }
    public static void increment(int n) {
        n++;
    }
}
• The value of the argument is copied, so no change to number!
```

This does the right thing

```java
• What do you expect this to print?
public class PassByValue {
    public static void main(String[] args) {
        int number = 100;
        number = increment(number);
        System.out.println("Number: " + number);
    }
    public static int increment(int n) {
        return ++n;
    }
}
• The correct value is returned, overwriting number.
```

Incorrect Swapping

```java
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!
```
Methods inside a class

• Order of writing methods is arbitrary
  – Generally constructors are written first
• What if two methods need to share data?
  – One subtask reads input and creates a string of words separated by white space
  – Another subtask checks each word in the string one at a time

First Solution

• Called method returns a value (data)
• Calling method uses the value (data)
• Example:
  
  ```java
  public static void main(String[] args) {
    String wordList = readInput();
    processWords(wordList);
  }
  ```

Second Solution

• Use instance variables
  – Define `String wordList;` as an instance variable in the class
  – Any method of a class can access its variables
    • `readInput()` can create and write the string
    • `processWords()` can access (read) the same string
  – Of course neither method can be `static` since the instance variable is not static!

Encapsulation

• Consider example of driving a car
  – We see and use break pedal, accelerator pedal, steering wheel – know `what` they do
  – We do not see mechanical details of `how` they do their jobs
• Encapsulation divides class definition into
  – Class `interface`
  – Class `implementation`
Interface/Implementation

- **A class interface**
  - Tells *what* the class does
  - Gives headings for public methods and comments about them
- **A class implementation**
  - Contains private variables
  - Includes definitions of public and private methods

Encapsulation/Interfaces

A well encapsulated class definition:

Interfaces

- Class definition has comments on how to use class
- All instance variables in the class declared as private.
- Provide public accessor methods to retrieve data
- Provide public mutator methods to manipulating data
- Add comments before each public method heading that fully specifies how to use method.
- Write comments within class definition to describe implementation details.
- Hide everything else: data structures, helper methods, other implementation details.

Interface Example

```java
public interface Trigonometry {
    public double sin(double angle);
    public double cos(double angle);
    public double tan(double angle);
}

public class Math implements Trigonometry {
    public double sin(double angle) { ... code ... }
    public double cos(double angle) { ... code ... }
    public double tan(double angle) { ... code ... }
}

Trigonometry trig = new Math();
```