Review: How To Write a Method

1. Write the method signature (name, return type, arguments)

   /*
   * Returns the contents of the web page identified
   * by urlString, must be a valid URL.
   * e.g. fetch("http://www.mit.edu")
   * returns the MIT home page as a string of HTML.
   */

   public static String fetch(String urlString) {

   ...
   }

2. Write a specification (a comment that defines what it returns, any side-effects, and assumptions about the arguments).

3. Write the method body so that it conforms to your specification. (Revise the signature or specification if you discover you can’t implement it!)

Getting Data from the Web

import java.net.URL;

   /*
   * Returns the contents of the web page identified
   * by urlString. e.g. fetch("http://www.mit.edu")
   * returns the MIT home page as a string of HTML.
   */

   public static String fetch(String urlString) {
       URL url = new URL(urlString);
       ...
   }

Today’s Topics
object-oriented programming in Java
- exceptions
- classes
- subclassing
Classes in Other Packages

Java classes are arranged in packages
- java.lang.String
- java.lang.Math
- java.net.URL

Import statements at top of Java file bring in the classes you need
- import java.net.URL;
- import java.net.*;
- java.lang.* package is imported automatically, so we don’t have to do this with String or Math, for example

Exceptions

Exceptions are abnormal return conditions from a method
- Instead of returning a value normally, the method throws an exception
- Exceptions usually indicate error conditions, but not necessarily
- Exceptions are objects. Usually just have a message, but can carry other data as well

Throwing an exception
- throw statement throws an exception object
  throw new MalformedURLException("bad URL:"); + urlString);
- throw is like return – the method immediately stops, but instead of returning a value, it propagates the exception

Two Ways To Deal With Exceptions

```java
public static String fetch(String urlString) {  
    try {  
        URL url = new URL(urlString);  
        // open a stream from the web server  
        // (defined in a few slides)  
        InputStream input = url.openStream();  
        InputStreamReader reader = new InputStreamReader(input);  
        // create a stream that appends data together into a String  
        StringWriter writer = new StringWriter();  
        // copy from the web server stream to the string stream  
        // (defined in a few slides)  
        copyStream(reader, writer);  
        // return the string we created  
        return writer.toString();  
    } catch (MalformedURLException e) {  
        System.out.println("Badly formed URL: ");  
        e.printStackTrace();  
        System.exit(0);  
    } catch (IOException e) {  
        // open a stream from the web server  
        // (defined in a few slides)  
        InputStream input = url.openStream();  
        InputStreamReader reader = new InputStreamReader(input);  
        // create a stream that appends data together into a String  
        StringWriter writer = new StringWriter();  
        // copy from the web server stream to the string stream  
        // (defined in a few slides)  
        copyStream(reader, writer);  
        // return the string we created  
        return writer.toString();  
    }
}
```

Getting Data from the Web

```java
public static String fetch(String urlString) throws MalformedURLException, IOException {  
    URL url = new URL(urlString);  
    // open a stream from the web server  
    InputStream input = url.openStream();  
    InputStreamReader reader = new InputStreamReader(input);  
    // create a stream that appends data together into a String  
    StringWriter writer = new StringWriter();  
    // copy from the web server stream to the string stream  
    // (defined in a few slides)  
    copyStream(reader, writer);  
    // return the string we created  
    return writer.toString();  
}```
Bytes vs. Chars

Byte is an 8-bit value
- Older programming systems used 7-bit (ASCII) or 8-bit character sets, which could represent at most 256 different characters
- The multilingual Web demands a lot more!
- But network connections and files are still generally represented as a sequence of 8-bit byte values
- java.io.InputStream and java.io.OutputStream are streams of bytes

Char is a 16-bit value
- Java characters are Unicode characters
- Unicode is an extension of ASCII, which has thousands of characters (including Latin alphabets, Greek, Cyrillic, Chinese/Japanese/Korean characters, symbols, accents, etc.)
- java.lang.String is a sequence of Unicode characters, and java.io.Reader and java.io.Writer are streams of Unicode characters
- If it’s human-readable text, use Unicode; if it’s binary data (like an image) use bytes

Try/Finally

```java
public static void copyStream(Reader from, Writer to)
    throws IOException {
    try {
        char[] buffer = new char[10000];
        // any size buffer would work, but bigger
        // performs better
        while (true) {
            int n = from.read(buffer);
            if (n == -1) break; // “from” stream is done
            to.write(buffer, 0, n);
        }
    }
    finally { // finally clause is run no matter how control leaves the try block—whether
        reader.close();
        writer.close();
    }
}
```

Reading and Writing Streams

```java
/*
 * Copies all data from the "from" stream to
 * the "to" stream, then closes both streams.
 * Throws IOException if any error occurs.
 */
public static void copyStream(Reader from, Writer to)
    throws IOException {
    char[] buffer = new char[10000];
    // any size buffer would work, but bigger
    // performs better
    while (true) {
        int n = from.read(buffer);
        if (n == -1) break; // “from” stream is done
        to.write(buffer, 0, n);
    }
    reader.close();
    writer.close();
}
```

Overloading a Method

```java
public static String fetch(String urlString)
    throws MalformedURLException, IOException {
    URL url = new URL(urlString);
    return fetch(url);
}
```

```java
public static String fetch(URL url)
    throws IOException {
    // open a connection to the web server
    InputStream input = url.openStream();
    InputStreamReader reader = new InputStreamReader(input);
    ...}
```

```java
public static String fetch(String urlString)
    throws MalformedURLException, IOException {
    URL url = new URL(urlString);
    return fetch(url);
}
```

```java
public static String fetch(URL url)
    throws IOException {
    // open a connection to the web server
    InputStream input = url.openStream();
    InputStreamReader reader = new InputStreamReader(input);
    ...}
```

It's important to close the streams to mark the end of the stream and free up resources. But will the streams always be closed in this code?

Java automatically chooses which overloaded method to call based on the types of the arguments you give it

```java
fetch("http://www.mit.edu");
fetch(new URL("http://www.mit.edu");
```
A Class Representing Web Pages

```java
public class Page {
    private URL url;
    private String content;

    public Page(String urlString) throws MalformedURLException {
        this.url = new URL(urlString);
    }

    public URL getURL() {
        return this.url;
    }

    public String getContent() {
        return this.content;
    }
}
```

**Fields are variables stored in the object.**

**Constructors create new objects.**

**Methods are functions that act on an object.**

**This refers to the object itself in a method or constructor.**

---

Access Control

- **Public** can be used anywhere in the program.
- **Private** can be used only in this class.

**Access control provides greater safety.**

We want Page to be immutable (never changes once created). What if its fields were public?

```java
public URL url;
```

Then it would be possible to change the field anywhere in the program, and Page would no longer be immutable.

Let's add a cache to the Page class.

```java
/* Returns the cached Page object for url, 
or null if no such Page in the cache. */
private static Page getPageFromCache(URL url) { ... }

/* Stores page in the cache. */
private static void putPageInCache(Page page) { ... }
```

---

Final

**Another way to control changes to a field.**

- Fields and variables marked `final` may not be reassigned after initialization.
- So Page could be kept immutable even if it's public.

```java
public final URL url;
```

It's good practice to use `final` for any variable that shouldn't be reassigned (even local variables).

```java
public static String fetch(String urlString) throws MalformedURLException {
    final URL url = new URL(urlString);
    final InputStream input = url.openStream();
    final InputStreamReader reader = new InputStreamReader(input);
    final StringWriter writer = new StringWriter();
    ...}
```

---

Caching Pages

**Web browsers store downloaded pages in a cache.**

- So that they don’t have to download the page each time it’s used.
- Let’s add a cache to the Page class.

```java
/* Returns the cached Page object for url, 
or null if no such Page in the cache. */
private static Page getPageFromCache(URL url) { ... }

/* Stores page in the cache. */
private static void putPageInCache(Page page) { ... }
```

Returning an invalid value (like null) is one way to signal an error condition. How else could we have designed this method to signal an error to its caller?
Static Fields and Methods

- Fields and methods declared static are associated with the class itself, rather than an individual object.
  - A static field has only one value for the whole program (rather than one value per object)
  - All objects of the class share that single copy of the static field
- A static method has no this object
- Static methods and fields are referenced using the class name (e.g. Web.fetch()) rather than an object variable
- Some classes are purely containers for static code (e.g. Hailstone, Web, java.lang.Math), and no objects of the class are ever constructed
- Fields and methods not declared static are called instance fields or methods
- static final is commonly used for constants, e.g.:
  
  public static final PI = 3.14159;

Implementing the Cache

```java
private static final Page[] cache = new Page[100];
private static int cachePointer = 0;
// index of next page to replace in the cache

private static Page getPageFromCache(URL url) {
    for (Page p : cache) {
        if (p != null && p.getURL().equals(url)) return p;
    }
    return null; // page not found
    // why might p be null?
    // what happens if we don't check?
}

private static void putPageInCache(Page page) {
    cache[cachePointer] = page;
    ++cachePointer;
    if (cachePointer >= cache.length) cachePointer = 0;
}
```

Summary

Exceptions

- Exceptions are abnormal returns from a method
- Exceptions can be caught or declared

Classes

- Members (fields, constructors, methods)
- Access control (public, protected, private)
- Static members
- Overloading