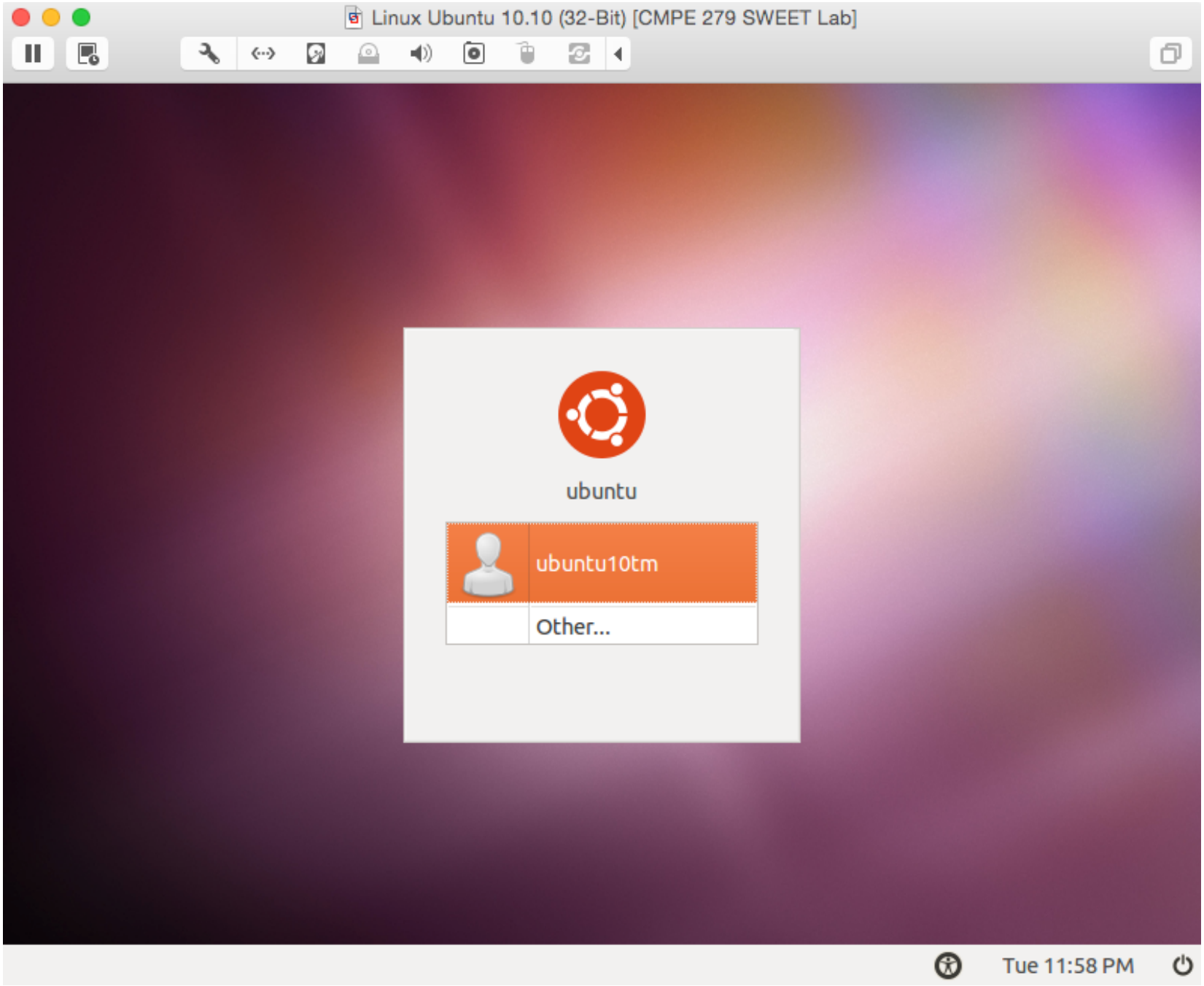
Software Security

Module 7 - Web Stack & Java Security

CMPE279
Software Security Technologies
San Jose State University

Lab VM Setup


Linux Ubuntu 10.10 (32-Bit) [CMPE 279 SWEET Lab]
ubuntu
ubuntu10tm
Other...
Tue 11:58 PM

Ubuntu Sweet VM

user login: user (ubuntu10tm)
user pass: 123456

root login: root
root pass: 123456

GitHub/SVN Checkout:

cd ~/CMPE279
svn co https://github.com/paulnguyen/cmpe279/trunk .

(to update, type "***svn update***" in CMPE279)

Tomcat 7

admin user: tomcat
password: tomcat

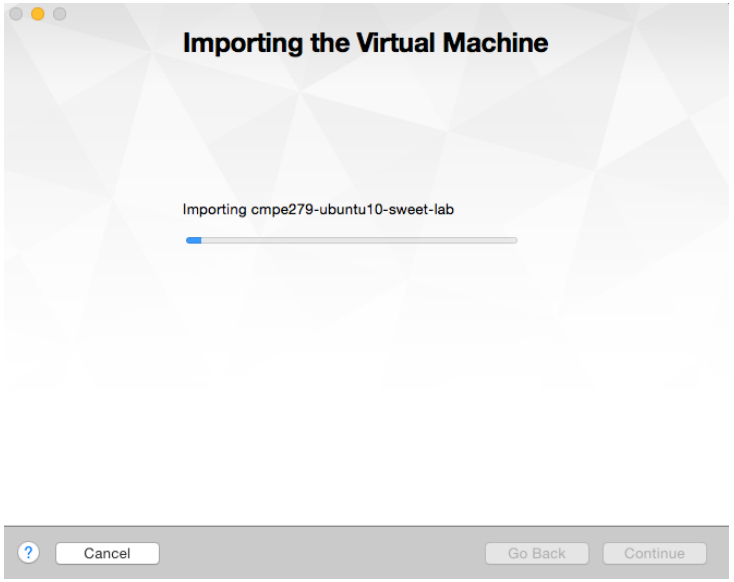
Tomcat 6 (WebGoat 5.3)

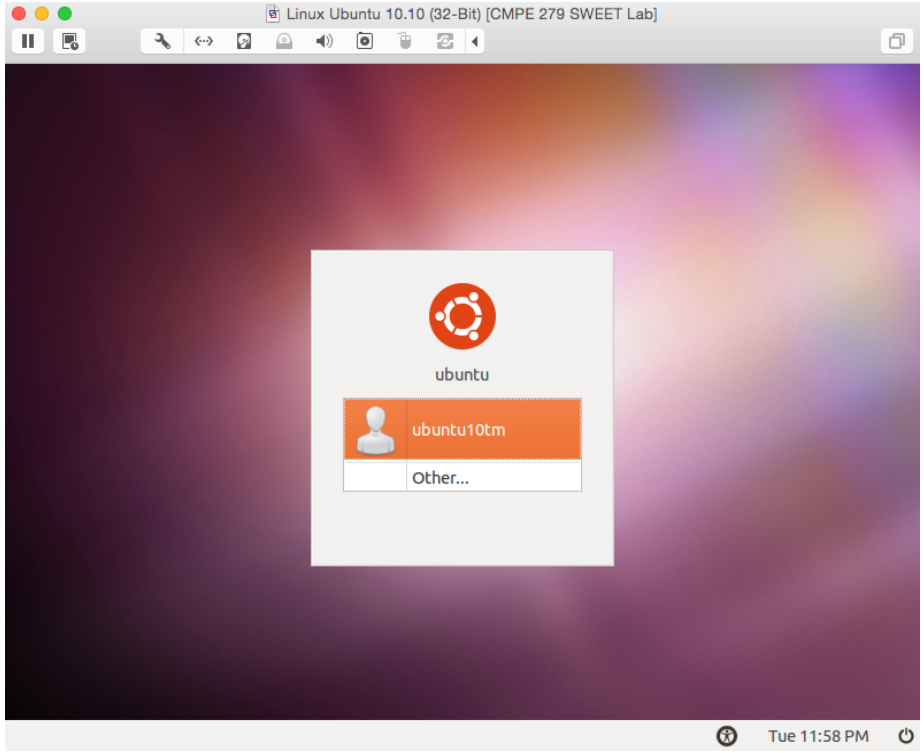
admin user: tomcat / tomcat
webgoat admin: webgoat / webgoat
webgoat basic: basic / basic
web goat guest: guest / guest

MySQL 5.1.49

root user: root / 123456

databases: badstoredb, mysql, test





Hello Java

with BlueJ

http://www.bluej.org/objects-first/

Primitive Types & Autoboxing

Primitive type	Wrapper type
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean

Primitive types

The following table lists all the primitive types of the Java language:

Type name	Description	Example literals		
Integer numbers				
`byte`	byte-sized integer (8 bit)	`24`	`-2`	
`short`	short integer (16 bit)	`137`	`-119`	
`int`	integer (32 bit)	`5409`	`-2003`	
`long`	long integer (64 bit)	`423266353L`	`55L`	
Real numbers				
`float`	single-precision floating point	`43.889F`		
`double`	double-precision floating point	`45.63`	`2.4e5`	
Other types				
`char`	a single character (16 bit)	`'m'`	`'?'`	`'\u00F6'`
`boolean`	a boolean value (true or false)	`true`	`false`	

Whenever a value of a primitive type is used in a context that requires an object type, the compiler uses autoboxing to automatically wrap the primitive-type value in an appropriate wrapper object. This means that primitive-type values can be added directly to a collection, for instance. The reverse operation – *unboxing* – is also performed automatically when a wrapper-type object is used in a context that requires a value of the corresponding primitive type.

Control Structures

```
if(expression) {
    statements
}
else {
    statements
}
```

```
switch (expression) {
    case value1:
    case value2:
    case value3:
       statements;
       break;
    case value4:
    case value5:
       statements;
       break;
    further cases omitted
    default:
       statements;
       break;
}
```

```
while(expression) {
      statements
}
```

```
do {
    statements
} while(expression);
```

```
try {
    statements
}
catch(exception-type name) {
     statements
}

finally {
    statements
}
```

```
for(variable-declaration: collection) {
    statements
}
```

```
for(String note: list) {
    System.out.println(note);
}
```

```
for(int i = 0; i < text.size(); i++) {
    System.out.println(text.get(i));
}
```

Class Definitions

```
/**
 * TicketMachine models a ticket machine that issues
 * flat-fare tickets.
 * The price of a ticket is specified via the constructor.
 * Instances will check to ensure that a user only enters
 * sensible amounts of money, and will only print a ticket
 * if enough money has been input.
 *
 * @author David J. Barnes and Michael Kolling
 * @version 2008.03.30
 */
public class TicketMachine
{
    // The price of a ticket from this machine.
    private int price;
    // The amount of money entered by a customer so far.
    private int balance;
    // The total amount of money collected by this machine.
    private int total;

    /**
     * Create a machine that issues tickets of the given price.
     */
    public TicketMachine(int ticketCost)
    {
        price = ticketCost;
        balance = 0;
        total = 0;
    }

    /**
     * @Return The price of a ticket.
     */
    public int getPrice()
    {
        return price;
    }

    /**
     * Return The amount of money already inserted for the
     * next ticket.
     */
    public int getBalance()
    {
        return balance;
    }
```

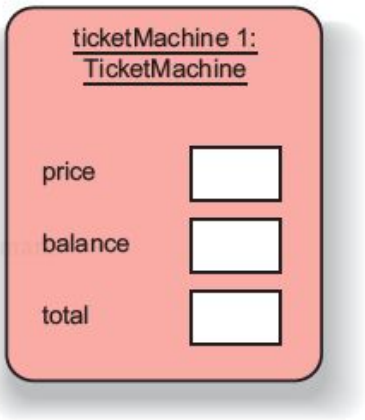
```
    /**
     * Receive an amount of money in cents from a customer.
     * Check that the amount is sensible.
     */
    public void insertMoney(int amount)
    {
        if(amount > 0) {
            balance = balance + amount;
        }
        else {
            System.out.println("Use a positive amount: " +
                               amount);
        }
    }

    /**
     * Print a ticket if enough money has been inserted, and
     * reduce the current balance by the ticket price. Print
     * an error message if more money is required.
     */
    public void printTicket()
    {
        if(balance >= price) {
            // Simulate the printing of a ticket.
            System.out.println("##################");
            System.out.println("# The BlueJ Line");
            System.out.println("# Ticket");
            System.out.println("# " + price + " cents.");
            System.out.println("##################");
            System.out.println();

            // Update the total collected with the price.
            total = total + price;
            // Reduce the balance by the prince.
            balance = balance - price;
        }
        else {
            System.out.println("You must insert at least: " +
                               (price - balance) + " more cents.");

        }
    }
}
```

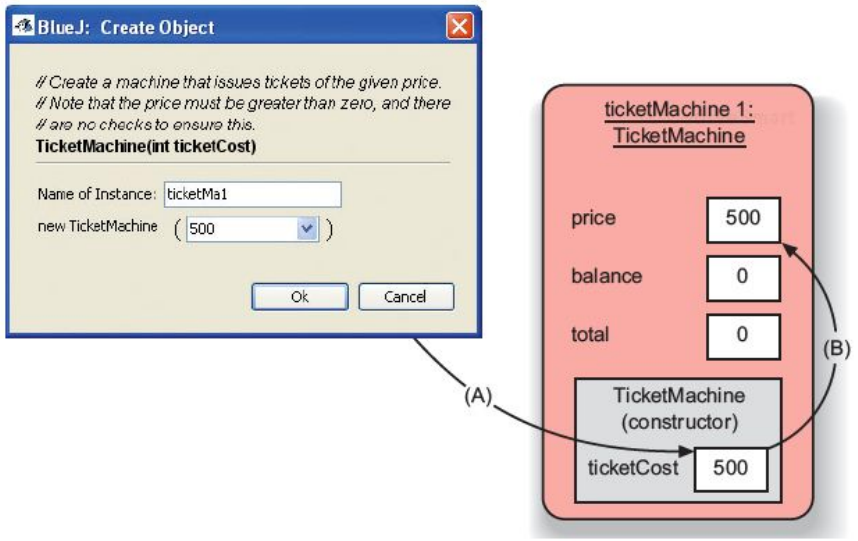
Fields & Constructors

```
public class ClassName
{
    Fields
    Constructors
    Methods
}
```

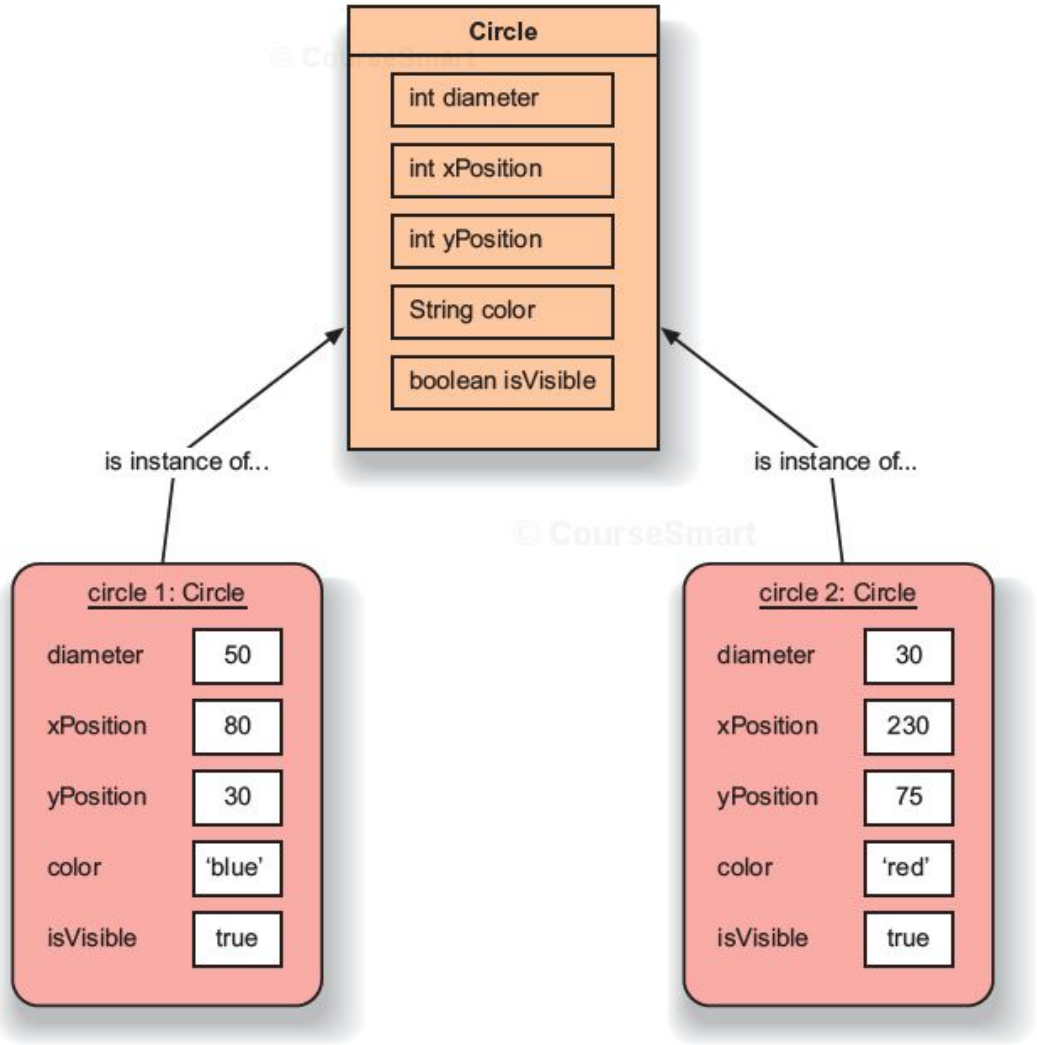
```
public class TicketMachine
{
    private int price;
    private int balance;
    private int total;

    Constructor and methods omitted.
}
```

```
public class TicketMachine
{
    Fields omitted.

    /**
     * Create a machine that issues tickets of the given price.
     * Note that the price must be greater than zero, and there
     * are no checks to ensure this.
     */
    public TicketMachine(int ticketCost)
    {
        price = ticketCost;
        balance = 0;
        total = 0;
    }

    Methods omitted.
}
```





Objects as Instances



Accessors

Concept:

Accessor methods return information about the state of an object.

```
public class TicketMachine
{
    Fields omitted.

    Constructor omitted.

    /**
     * Return the price of a ticket.
     */
    public int getPrice()
    {
        return price;
    }

    Remaining methods omitted.
}
```

Mutators

```
public class TicketMachine
{
    // The price of a ticket from this machine.
    private int price;
    // The amount of money entered by a customer so far.
    private int balance;
    // The total amount of money collected by this machine.
    private int total;
```

Concept:

Mutator methods change the state of an object.

```
/**
 * Receive an amount of money in cents from a customer.
 */
public void insertMoney(int amount)
{
    balance = balance + amount;
}
```

Printing to the Screen

Concept:

The method **System.out.println** prints its parameter to the text terminal.

```
/**
 * Print a ticket and reduce the
 * current balance to zero.
 */
public void printTicket()
{
    // Simulate the printing of a ticket.
    System.out.println("##################");
    System.out.println("# The BlueJ Line");
    System.out.println("# Ticket");
    System.out.println("# " + price + " cents.");
    System.out.println("##################");
    System.out.println();

    // Update the total collected with the balance.
    total = total + balance;
    // Clear the balance.
    balance = 0;
}
```

Abstraction

Concept:

Classes define types. A class name can be used as the type for a variable. Variables that have a class as their type can store objects of that class.

The clock example


```
public class NumberDisplay
{
    private int limit;
    private int value;

    Constructor and methods omitted.
}
```

```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;

    Constructor and methods omitted.
}
```

Concept:

Abstraction is the ability to ignore details of parts to focus attention on a higher level of a problem.

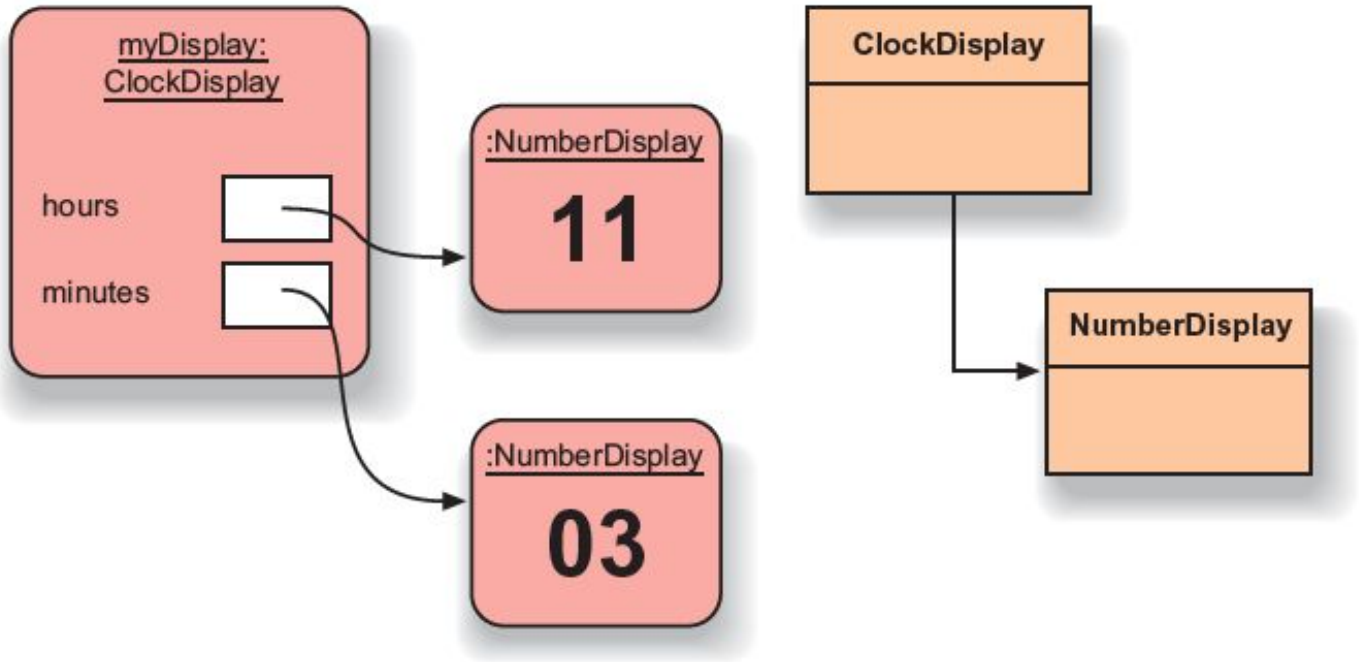
Concept:

Modularization is the process of dividing a whole into well-defined parts, which can be built and examined separately, and which interact in well-defined ways.

Classes versus Objects

Concept:

The **class diagram** shows the classes of an application and the relationships between them. It gives information about the source code. It presents the static view of a program.



The Clock Display

```
public class NumberDisplay
{
    private int limit;
    private int value;
```

```
    /**
     * Constructor for objects of class NumberDisplay
     */
    public NumberDisplay(int rollOverLimit)
    {
        limit = rollOverLimit;
        value = 0;
    }

    /**
     * Return the current value.
     */
    public int getValue()
    {
        return value;
    }

    /**
     * Set the value of the display to the new specified
     * value. If the new value is less than zero or over the
     * limit, do nothing.
     */
    public void setValue(int replacementValue)
    {
        if((replacementValue >= 0) &&
                (replacementValue < limit)) {
            value = replacementValue;
        }
    }
```

The clock example

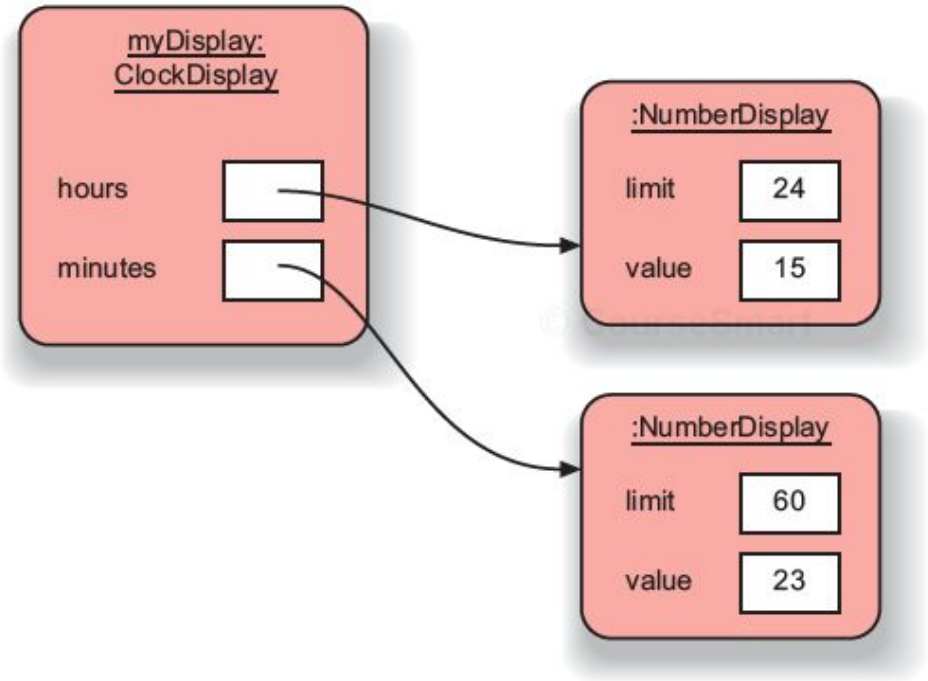

```
    /**
     * Increment the display value by one, rolling over to zero if
     * the limit is reached.
     */
    public void increment()
    {
        value = (value + 1) % limit;
    }
```

```
    /**
     * Return the display value (that is, the current value
     * as a two-digit String. If the value is less than ten,
     * it will be padded with a leading zero).
     */
    public String getDisplayValue()
    {
        if(value < 10) {
            return "0" + value;
        }
        else {
            return "" + value;
        }
    }
```

Objects Creating Objects

Concept:

Object creation. Objects can **create** other **objects** using the new operator.




```
public class ClockDisplay
{
    private NumberDisplay hours;
    private NumberDisplay minutes;

    Remaining fields omitted.

    public ClockDisplay()
    {
        hours = new NumberDisplay(24);
        minutes = new NumberDisplay(60);
        updateDisplay();
    }

    Methods omitted.
}
```

Overloading

Concept:

Overloading. A class may contain more than one constructor, or more than one method of the same name, as long as each has a distinctive set of parameter types.

```
new ClockDisplay()
new ClockDisplay(hour, minute)
```

Method Calls

Concept:

Methods can call methods of other objects using dot notation. This is called an **external method call**.

object.method()

```
public void timeTick()
{
    minutes.increment();
    if(minutes.getValue() == 0) { // it just rolled over!
        hours.increment();
    }
    updateDisplay();
}
```

Concept:

Methods can call other methods of the same class as part of their implementation. This is called an **internal method call**.

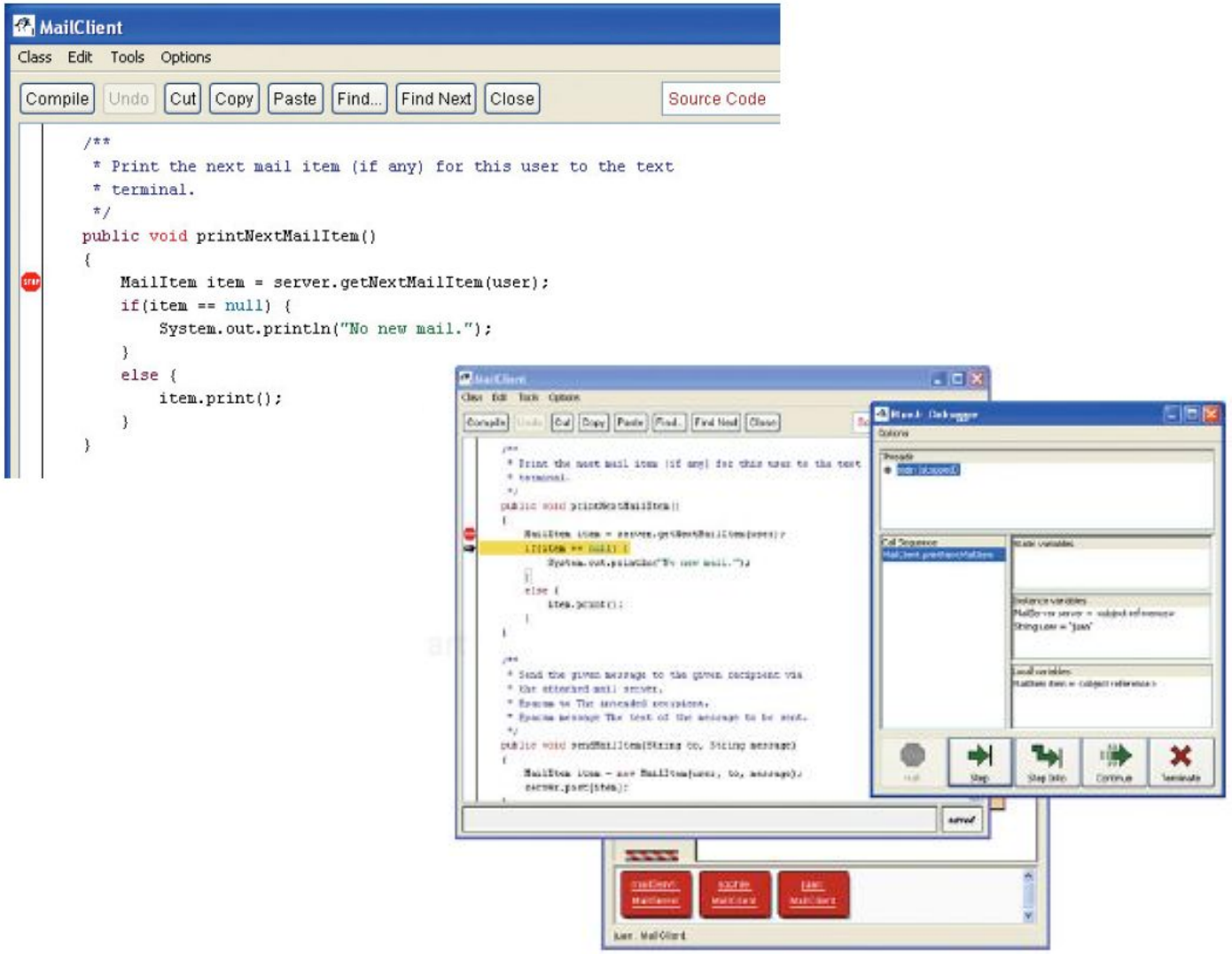
Implied “this” reference. i.e. this.updateDisplay()

“this” keyword

```
public class MailItem
{
    // The sender of the item.
    private String from;
    // The intended recipient.
    private String to;
    // The text of the message.
    private String message;

    /**
     * Create a mail item from sender to the given recipient,
     * containing the given message.
     * @param from    The sender of this item.
     * @param to      The intended recipient of this item.
     * @param message The text of the message to be sent.
     */
    public MailItem(String from, String to, String message)
    {
        this.from = from;
        this.to = to;
        this.message = message;
    }

    Methods omitted.
}
```

Debugging in BlueJ

Set Breakpoint



Using Java SDK

http://download.oracle.com/javase/7/docs/api/

```
import java.util.ArrayList;

/**
 * A class to maintain an arbitrarily long list of notes.
 * Notes are numbered for external reference by a human user.
 * In this version, note numbers start at 0.
 *
 * @author David J. Barnes and Michael Kölling.
 * @version 2008.03.30
 */
public class Notebook
{
    // Storage for an arbitrary number of notes.
    private ArrayList<String> notes;

    /**
     * Perform any initialization that is required for the
     * notebook.
     */
    public Notebook()
    {
        notes = new ArrayList<String>();
    }

    /**
     * Store a new note into the notebook.
     * @param note The note to be stored.
     */
    public void storeNote(String note)
    {
        notes.add(note);
    }

    /**
     * @return The number of notes currently in the notebook.
     */
    public int numberOfNotes()
    {
        return notes.size();
    }
```

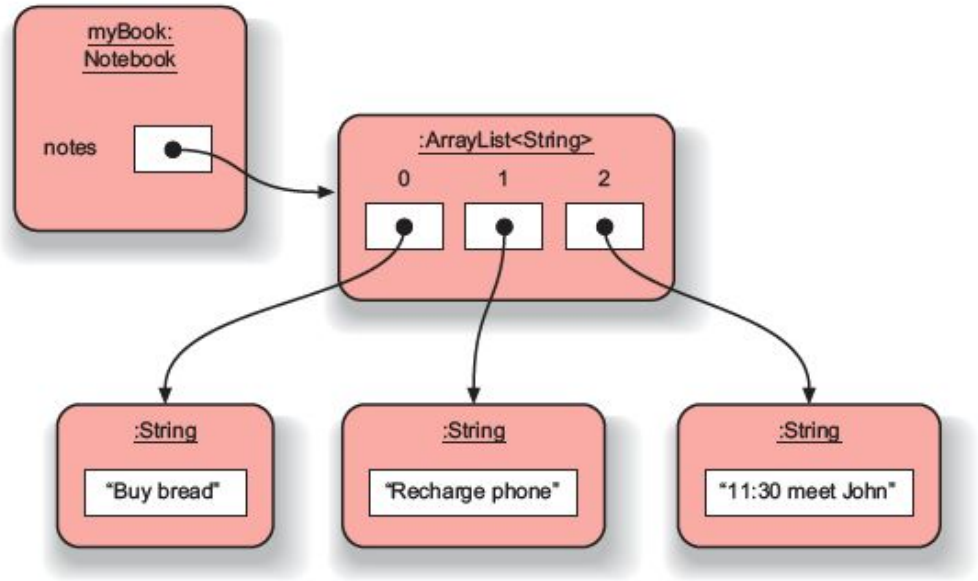
Import Class Definition

Now, can say:
ArrayList
instead of
java.util.ArrayList

```
    /**
     * Show a note.
     * @param noteNumber The number of the note to be shown.
     */
    public void showNote(int noteNumber)
    {
        if(noteNumber < 0) {
            // This is not a valid note number, so do nothing.
        }
        else if(noteNumber < numberOfNotes()) {
            // This is a valid note number, so we can print it.
            System.out.println(notes.get(noteNumber));
        }
        else {
            // This is not a valid note number, so do nothing.
        }
    }
}
```

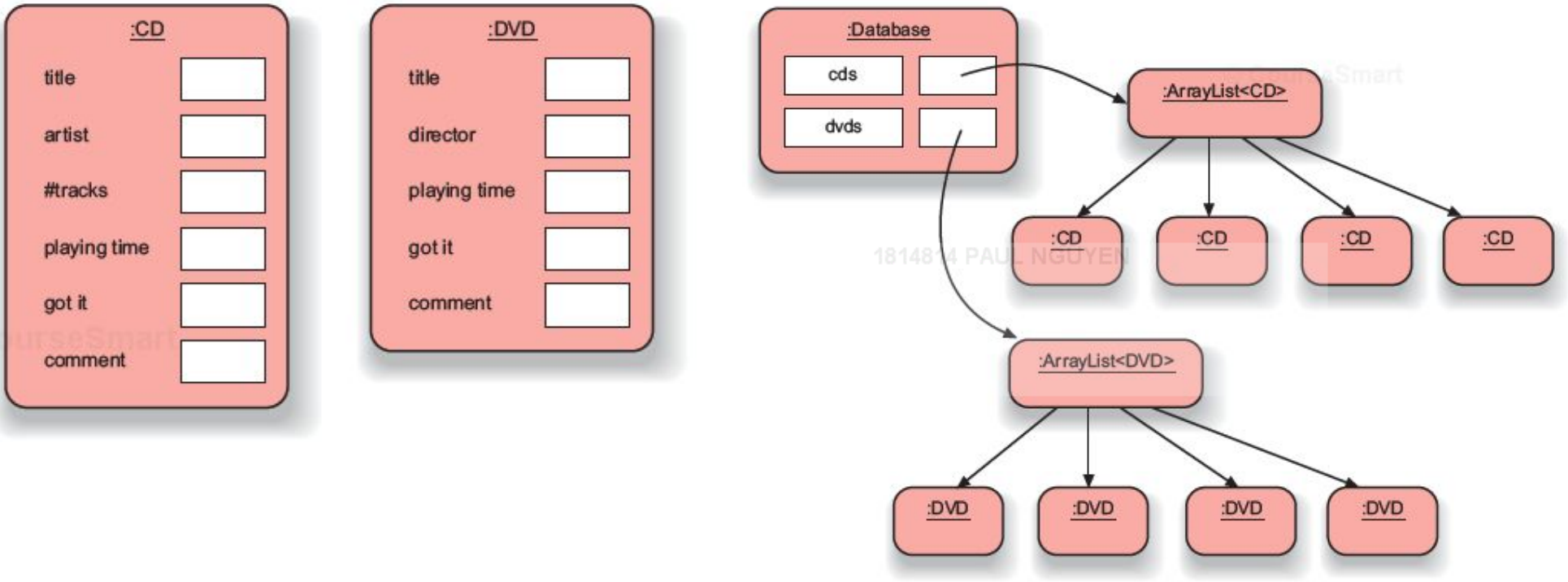
Adding & Removing from a List

```
    /**
     * Store a new note into the notebook.
     * @param note The note to be stored.
     */
    public void storeNote(String note)
    {
        notes.add(note);
    }
```

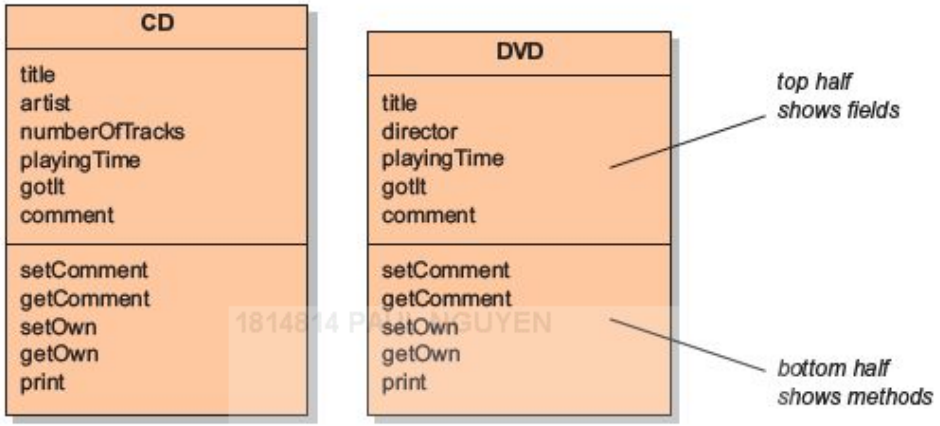



```
public void removeNote(int noteNumber)
{
    if(noteNumber < 0) {
        // This is not a valid note number, so do nothing.
    }
    else if(noteNumber < numberOfNotes()) {
        // This is a valid note number, so we can remove it.
        notes.remove(noteNumber);
    }
    else {
        // This is not a valid note number, so do nothing.
    }
}
```

Object Structure

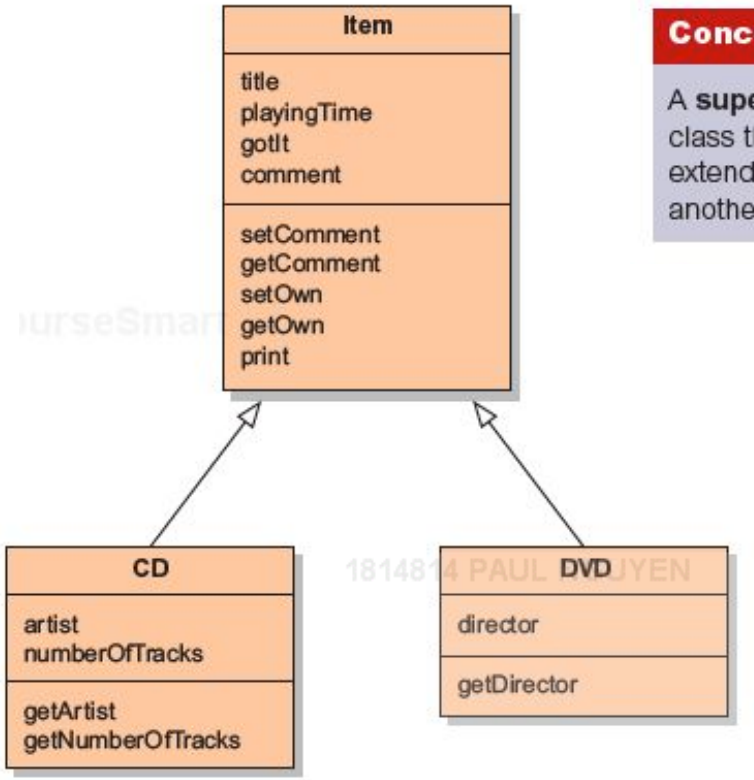


Using Inheritance



Concept:

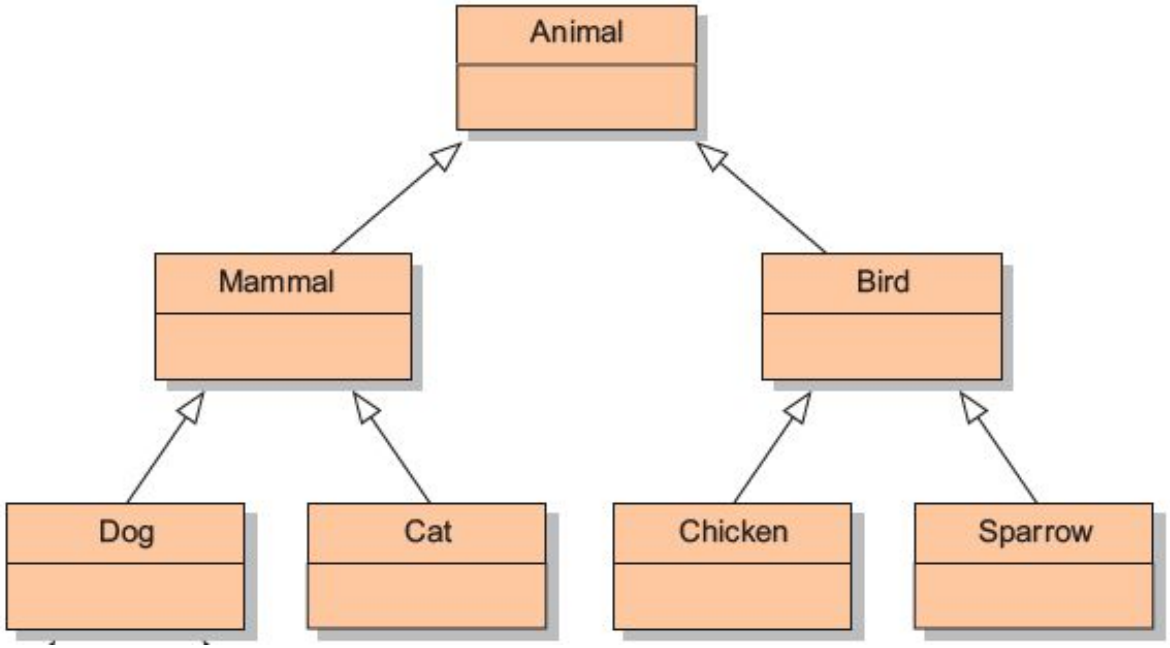
A **subclass** is a class that extends (inherits from) another class. It inherits all fields and methods from its superclass.



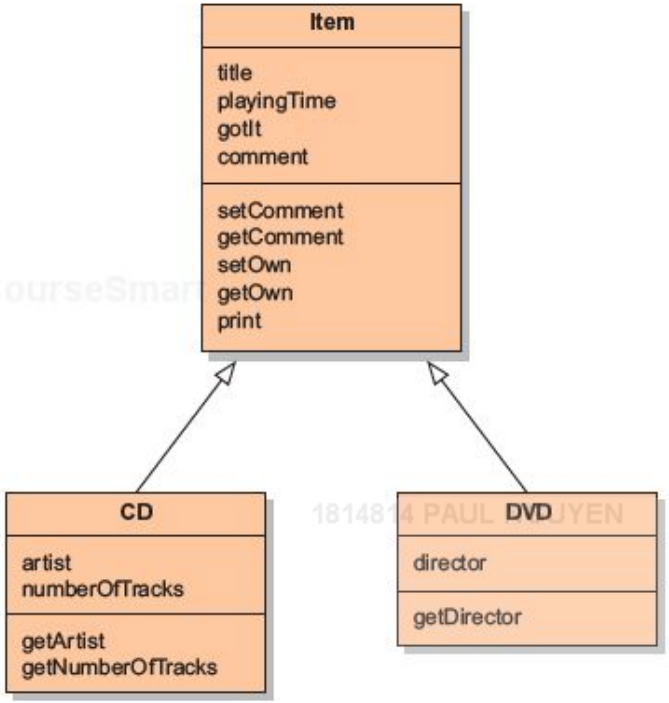
Concept:

A **superclass** is a class that is extended by another class.

Object Hierarchies in the real world



Inheritance in Java




```
public class Item
{
    private String title;
    private int playingTime;
    private boolean gotIt;
    private String comment;

    // constructors and methods omitted.
}
```

```
public class CD extends Item
{
    private String artist;
    private int numberOfTracks;

    // constructors and methods omitted.
}
```

```
public class DVD extends Item
{
    private String director;

    // constructors and methods omitted.
}
```

Abstract Classes

Concept:

An **abstract method** definition consists of a method signature without a method body. It is marked with the key word `abstract`.

Concept:

An **abstract class** is a class that is not intended for creating instances. Its purpose is to serve as a superclass for other classes. Abstract classes may contain abstract methods.

```
public abstract class Animal
{
    // fields omitted

    /**
     * Make this animal act – that is: make it do whatever
     * it wants/needs to do.
     * @param newAnimals A list to add newly born animals to.
     */
    abstract public void act(List<Animal> newAnimals);

    // other methods omitted
}
```

Concept:

Abstract subclass. For a subclass of an abstract class to become concrete, it must provide implementations for all inherited abstract methods. Otherwise the subclass will itself be abstract.

Interfaces

Concept:

A Java **interface** is a specification of a type (in the form of a type name and a set of methods) that does not define any implementation for the methods.

Note: Java does not support multiple inheritance

```
public class Fox extends Animal implements Drawable
{
    // Body of class omitted.
}
```

```
/**
 * The interface to be extended by any class wishing
 * to participate in the simulation.
 */
public interface Actor
{
    /**
     * Perform the actor's regular behavior.
     * @param newActors A list for storing newly created actors.
     */
    void act(List<Actor> newActors);

    /**
     * Is the actor still active?
     * @return true if still active, false if not.
     */
    boolean isActive();
}
```

```
public class Hunter implements Actor, Drawable
{
    // Body of class omitted.
}
```

Important Java Libraries http://download.oracle.com/javase/7/docs/api/

package `java.lang` – Summary of the most important classes	
interface `Comparable`	Implementation of this interface allows comparison and ordering of objects from the implementing class. Static utility methods such as `Arrays.sort` and `Collections.sort` can then provide efficient sorting in such cases, for instance.
class `Math`	`Math` is a class containing only static fields and methods. Values for the mathematical constants `e` and π are defined here, along with trigonometric functions, and others such as `abs`, `min`, `max`, and `sqrt`.
class `Object`	All classes have `Object` as a superclass at the root of their class hierarchy. From it all objects inherit default implementations for important methods such as `equals` and `toString`. Other significant methods defined by this class are `clone` and `hashCode`.
class `String`	Strings are an important feature of many applications, and they receive special treatment in Java. Key methods of the `String` class are `charAt`, `equals`, `indexOf`, `length`, `split`, and `substring`. Strings are immutable objects, so methods such as `trim` that appear to be mutators actually return a new `String` object representing the result of the operation.
class `StringBuilder`	The `StringBuilder` class offers an efficient alternative to `String` when it is required to build up a string from a number of components: e.g., via concatenation. Its key methods are `append`, `insert`, and `toString`.

package `java.util` – Summary of the most important classes and interfaces

interface `Collection`	This interface provides the core set of methods for most of the collection-based classes defined in the `java.util` package, such as `ArrayList`, `HashSet`, and `LinkedList`. It defines signatures for the `add`, `clear`, `iterator`, `remove`, and `size` methods.
interface `Iterator`	`Iterator` defines a simple and consistent interface for iterating over the contents of a collection. Its three methods are `hasNext`, `next`, and `remove`.
interface `List`	`List` is an extension of the `Collection` interface,and provides a means to impose a sequence on the selection. As such, many of its methods take an index parameter: for instance, `add`, `get`, `remove`, and `set`. Classes such as `ArrayList` and `LinkedList` implement `List`.
interface `Map`	The `Map` interface offers an alternative to list-based collections by supporting the idea of associating each object in a collection with a *key* value. Objects are added and retrieved via its `put` and `get` methods. Note that a `Map` does not return an `Iterator`, but its `keySet` method returns a `Set` of the keys, and its `values` method returns a `Collection` of the objects in the map.
interface `Set`	`Set` extends the `Collection` interface with the intention of mandating that a collection contain no duplicate elements. It is worth pointing out that, because it is an interface, `Set` has no actual implication to enforce this restriction. This means that `Set` is actually provided as a marker interface to enable collection implementers to indicate that their classes fulfill this particular restriction.
class `ArrayList`	An implementation of the `List` interface that uses an array to provide efficient direct access via integer indices to the objects it stores. If objects are added or removed from anywhere other than the last position in the list, then following items have to be moved to make space or close the gap. Key methods are `add`, `get`, `iterator`, `remove`, and `size`.
class `Collections`	`Collections` is a collecting point for static methods that are used to manipulate collections. Key methods are `binarySearch`, `fill`, and `sort`.
class `HashMap`	`HashMap` is an implementation of the `Map` interface. Key methods are `get`, `put`, `remove`, and `size`. Iteration over a `HashMap` is usually a two-stage process: obtain the set of keys via its `keySet` method, and then iterate over the keys.
class `HashSet`	`HashSet` is a hash-based implementation of the `Set` interface. It is closer in usage to a `Collection` than to a `HashMap`. Key methods are `add`, `remove`, and `size`.
class `LinkedList`	`LinkedList` is an implementation of the `List` interface that uses an internal linked structure to store objects. Direct access to the ends of the list is efficient, but access to individual objects via an index is less efficient than with an `ArrayList`. On the other hand, adding objects or removing them from within the list requires no shifting of existing objects. Key methods are `add`, `getFirst`, `getLast`, `iterator`, `removeFirst`, `removeLast`, and `size`.
class `Random`	The `Random` class supports generation of pseudo-random values – typically random numbers. The sequence of numbers generated is determined by a *seed value*, which may be passed to a constructor or set via a call to `setSeed`. Two `Random` objects starting from the same seed will return the same sequence of values to identical calls. Key methods are `nextBoolean`, `nextDouble`, `nextInt`, and `setSeed`.
class `Scanner`	The `Scanner` class provides a way to read and parse input. It is often used to read input from the keyboard. Key methods are `next` and `hasNext`.

package `java.io` – Summary of the most important classes and interfaces

interface `Serializable`	The `Serializable` interface is an empty interface requiring no code to be written in an implementing class. Classes implement this interface in order to be able to participate in the serialization process. `Serializable` objects may be written and read as a whole to and from sources of output and input. This makes storage and retrieval of persistent data a relatively simple process in Java. See the `ObjectInputStream` and `ObjectOutputStream` classes for further information.
class `BufferedReader`	`BufferedReader` is a class that provides buffered character-based access to a source of input. Buffered input is often more efficient than unbuffered, particularly if the source of input is in the external file system. Because it buffers input, it is able to offer a `readLine` method that is not available in most other input classes. Key methods are `close`, `read`, and `readLine`.
class `BufferedWriter`	`BufferedWriter` is a class that provides buffered character-based output. Buffered output is often more efficient than unbuffered, particularly if the destination of the output is in the external file system. Key methods are `close`, `flush`, and `write`.
class `File`	The `File` class provides an object representation for files and folders (directories) in an external file system. Methods exist to indicate whether a file is readable and/or writeable, and whether it is a file or a folder. A `File` object can be created for a non-existent file, which may be a first step in creating a physical file on the file system. Key methods are `canRead`, `canWrite`, `createNewFile`, `createTempFile`, `getName`, `getParent`, `getPath`, `isDirectory`, `isFile`, and `listFiles`.
class `FileReader`	The `FileReader` class is used to open an external file ready for reading its contents as characters. A `FileReader` object is often passed to the constructor of another reader class (such as a `BufferedReader`) rather than being used directly. Key methods are `close` and `read`.
class `FileWriter`	The `FileWriter` class is used to open an external file ready for writing character-based data. Pairs of constructors determine whether an existing file will be appended or its existing contents discarded. A `FileWriter` object is often passed to the constructor of another `Writer` class (such as a `BufferedWriter`) rather than being used directly. Key methods are `close`, `flush`, and `write`.
class `IOException`	`IOException` is a checked exception class that is at the root of the exception hierarchy of most input/output exceptions.

package `java.net` – Summary of the most important classes	
class URL	The URL class represents a Uniform Resource Locator: in other words, it provides a way to describe the location of something on the Internet. In fact, it can also be used to describe the location of something on a local file system. We have included it here because classes from the `java.io` and `javax.swing` packages often use URL objects. Key methods are `getContent`, `getFile`, `getHost`, `getPath`, and `openStream`.

Other important packages

Other important packages are

```
java.awt
java.awt.event
javax.swing
javax.swing.event
```

These are used extensively when writing graphical user interfaces (GUIs), and they contain many useful classes that a GUI programmer should become familiar with.

Data Abstraction

From C to Java (and Beyond?)

Evolution of Abstraction

- **Code Lines**
  - classical “goto” problems
- **Functions**
  - poor encapsulation of related functions and data
- **Objects**
  - concerns that cross-cut object hierarchies not well modularized
- **Aspects**
  - modularizes cross-cutting concerns
  - return of the “goto”

Examples of Cross-Cutting Concerns

- *Logging*
- *Tracing*
- *Error handling*
- *Performance tuning*
- *Design patterns*
- *Contract Enforcement*

The Gumball Machine

Implementation in C (version 1) - Typical Solution

main.c

```c
#include <stdio.h>

int has_quarter = 0 ;
int num_gumballs = 1 ;

void insert_quarter( int coin )
{
    if ( coin == 25 )
       has_quarter = 1 ;
    else
        has_quarter = 0 ;
}

void turn_crank()
{
    if ( has_quarter )
    {
        if ( num_gumballs > 0 )
        {
            num_gumballs-- ;
            has_quarter = 0 ;
            printf( "Thanks for your quarter.  Gumball Ejected!\n" ) ;
        }
        else
        {
            printf( "No More Gumballs!  Sorry, can't return your quarter.\n" ) ;
        }
    }
    else
    {
        printf( "Please insert a quarter\n" ) ;
    }
}

int main(int argc, char **argv)
{
    printf("Simple Gumball Machine - Version 1\n");
    insert_quarter( 25 ) ;
    turn_crank() ;
    insert_quarter( 25 ) ;
    turn_crank() ;
    insert_quarter( 10 ) ;
    turn_crank() ;
    return 0;
}
```

Problems:

1. Global variables are not a good practice! Why?
2. Can only have one gumball machine at a time.
3. What about a 50¢ gumball machine?
4. How about the "nice" gumball machine that returns coins?

Implementation in C (version 2) - Abstract Data Type

gumball.h

```
1
2   typedef struct
3   {
4       int num_gumballs ;
5       int has_quarter ;
6   } GUMBALL ;
7
8   extern void init_gumball( GUMBALL *ptr, int size ) ;
9   extern void turn_crank( GUMBALL *ptr );
10  extern void insert_quarter( GUMBALL *ptr, int coin );
```

*main.c

```
1   #include <stdio.h>
2   #include "gumball.h"
3
4   int main(int argc, char **argv)
5   {
6       GUMBALL m1[1] ;
7       GUMBALL m2[1] ;
8
9       /* init gumball machines */
10      init_gumball( m1, 1 ) ;
11      init_gumball( m2, 10 ) ;
12
13      printf("Simple Gumball Machine - Version 2\n");
14
15      insert_quarter( m1, 25 ) ;
16      turn_crank( m1 ) ;
17      insert_quarter( m1, 25 ) ;
18      turn_crank( m1 ) ;
19      insert_quarter( m1, 10 ) ;
20      turn_crank( m1 ) ;
21
22      insert_quarter( m2, 25 ) ;
23      turn_crank( m2 ) ;
24      insert_quarter( m2, 25 ) ;
25      turn_crank( m2 ) ;
26      insert_quarter( m1, 10 ) ;
27      turn_crank( m2 ) ;
28
29      return 0;
30  }
```

gumball.c

```
1
2   #include <stdio.h>
3   #include "gumball.h"
4
5   void init_gumball( GUMBALL *ptr, int size )
6   {
7       ptr->num_gumballs = size ;
8       ptr->has_quarter = 0 ;
9   }
10
11  void turn_crank( GUMBALL *ptr )
12  {
13      if ( ptr->has_quarter )
14      {
15          if ( ptr->num_gumballs > 0 )
16          {
17              ptr->num_gumballs-- ;
18              ptr->has_quarter = 0 ;
19              printf( "Thanks for your quarter.  Gumball Ejected!\n" ) ;
20          }
21          else
22          {
23              printf( "No More Gumballs!  Sorry, can't return your quarter.\n" ) ;
24          }
25      }
26      else
27      {
28          printf( "Please insert a quarter\n" ) ;
29      }
30  }
31
32  void insert_quarter( GUMBALL *ptr, int coin )
33  {
34      if ( coin == 25 )
35          ptr->has_quarter = 1 ;
36      else
37          ptr->has_quarter = 0 ;
38  }
```

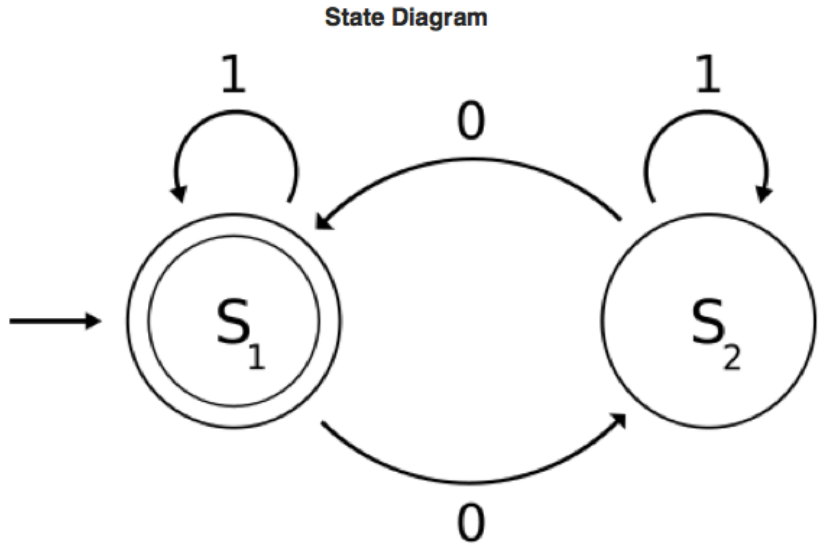
ADT --> think of the GB Machine in terms of its operations. (insert quarter, turn crank)
Data Encapsulation --> hide implementation details. Not supported in C Language

Implementation in C (version 3) - State Machine Design

```
typedef struct
{
    int num_gumballs ;
    int has_quarter ;
    int current_state ;
    /*  0 = OUT_OF_GUMBALL,
        1 = NO_QTR,
        2 = HAS_QTR,
        3 = EJECT_GUMBALL */
} GUMBALL ;

extern void init_gumball( GUMBALL *ptr, int size ) ;
extern void turn_crank( GUMBALL *ptr );
extern void insert_quarter( GUMBALL *ptr );
extern void eject_quarter( GUMBALL *ptr );
```

State Transition Table

Input State	1	0
S_1	S_1	S_2
S_2	S_2	S_1



http://en.wikipedia.org/wiki/State_transition_table

Implementation in C (version 3) - State Machine Design

```
void OUT_OF_GUMBALL( GUMBALL *ptr )
{
    printf( "No More Gumballs! \n" ) ;
    ptr->current_state = 0 ; /* No Gumballs! */
}

void NO_QTR( GUMBALL *ptr )
{
    printf( "No Quarter Inserted! \n" ) ;
    ptr->current_state = 2 ; /* No Quarter! */
}

void HAS_QTR( GUMBALL *ptr )
{
    printf( "Quarter Inserted! \n" ) ;
    ptr->current_state = 1 ; /* Has Quarter! */
    ptr->has_quarter = 1 ;
}

void EJECT_GUMBALL( GUMBALL *ptr )
{
    if( ptr->num_gumballs>0 )
    {
        printf( "Your Gumball has been ejected!\n" ) ;
        ptr->has_quarter = 0 ;
        ptr->num_gumballs-- ;
    }
    if( ptr->num_gumballs <= 0 )
        ptr->current_state = 0 ; /* Out of Gumballs! */
    else
        ptr->current_state = 2 ; /* No Quarter */
}

void (*machine[3][4]) (GUMBALL *ptr) = {
                    /* OUT_OF_GUMBALL,      HAS_QTR,          NO_QTR,      EJECT_GUMBALL   */
    /* crank */     {OUT_OF_GUMBALL,        EJECT_GUMBALL,    NO_QTR,      0             },
    /* insert qtr */{OUT_OF_GUMBALL,        HAS_QTR,          HAS_QTR,     0             },
    /* eject qtr */ {OUT_OF_GUMBALL,        NO_QTR,           NO_QTR,      0             }
} ;
```

```
void init_gumball( GUMBALL *ptr, int size )
{
    ptr->num_gumballs = size ;
    ptr->has_quarter = 0 ;
    ptr->current_state = 2 ; /* No Quarter */
}

void turn_crank( GUMBALL *ptr )
{
    machine[0][ptr->current_state](ptr) ;
}

void insert_quarter( GUMBALL *ptr )
{
    machine[1][ptr->current_state](ptr) ;
}

void eject_quarter( GUMBALL *ptr )
{
    machine[2][ptr->current_state](ptr) ;
}
```

Implementation in C (version 3) - State Machine Design

```c
#include <stdio.h>
#include "gumball.h"

int main(int argc, char **argv)
{
    GUMBALL m1[1] ;
    GUMBALL m2[1] ;

    /* init gumball machines */
    init_gumball( m1, 1 ) ;
    init_gumball( m2, 10 ) ;

    printf("Simple Gumball Machine - Version 3\n");

    insert_quarter( m1 ) ;
    turn_crank( m1 ) ;
    insert_quarter( m1 ) ;
    turn_crank( m1 ) ;
    insert_quarter( m1 ) ;
    turn_crank( m1 ) ;

    insert_quarter( m2 ) ;
    turn_crank( m2 ) ;
    turn_crank( m2 ) ;
    insert_quarter( m2 ) ;
    eject_quarter( m2 ) ;

    return 0;
}
```

Alternative way to support variations. More robust support for changing requirements

Implementation in Java (BlueJ)

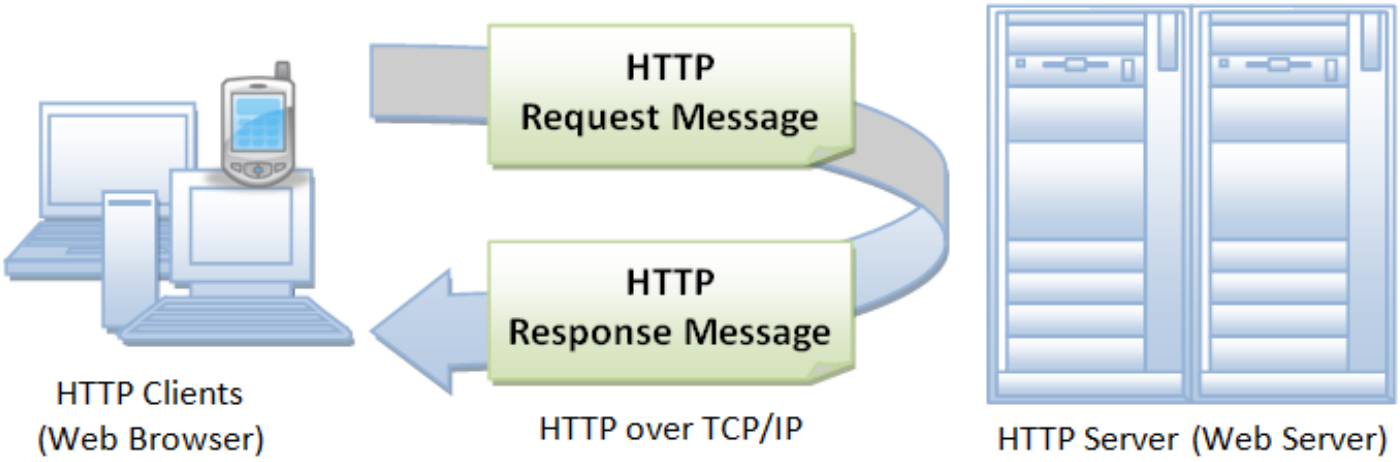
```
1
2  public class GumballMachine
3  {
4
5      private int num_gumballs;
6      private boolean has_quarter;
7
8      public GumballMachine( int size )
9      {
10         // initialise instance variables
11         this.num_gumballs = size;
12         this.has_quarter = false;
13     }
14
15     public void insertQuarter(int coin)
16     {
17         if ( coin == 25 )
18             this.has_quarter = true ;
19         else
20             this.has_quarter = false ;
21     }
22
23     public void turnCrank()
24     {
25         if ( this.has_quarter )
26         {
27             if ( this.num_gumballs > 0 )
28             {
29                 this.num_gumballs-- ;
30                 this.has_quarter = false ;
31                 System.out.println( "Thanks for your quarter.  Gumball Ejected!" ) ;
32             }
33             else
34             {
35                 System.out.println( "No More Gumballs!  Sorry, can't return your quarter." ) ;
36             }
37         }
38         else
39         {
40             System.out.println( "Please insert a quarter" ) ;
41         }
42     }
43 }
44
```

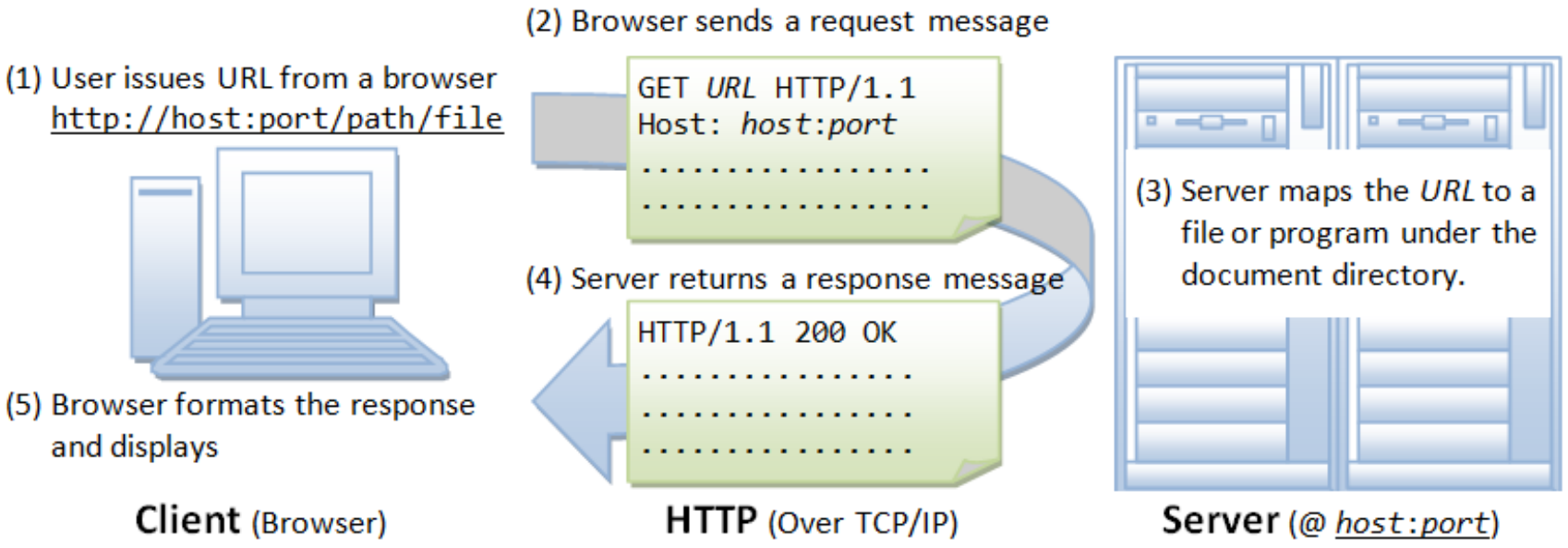
Native support for Encapsulation in OO Languages (like Java)

HTTP

The Protocol of the Web

https://www3.ntu.edu.sg/home/ehchua/programming/index.html


HTTP
Request Message
HTTP
Response Message
HTTP Clients
(Web Browser)
HTTP over TCP/IP
HTTP Server (Web Server)


(1) User issues URL from a browser
http://host:port/path/file
(2) Browser sends a request message
GET URL HTTP/1.1
Host: host:port
(3) Server maps the URL to a file or program under the document directory.
(4) Server returns a response message
HTTP/1.1 200 OK
(5) Browser formats the response and displays
Client (Browser)
HTTP (Over TCP/IP)
Server (@ host:port)

Uniform Resource Locator (URL)

A URL (Uniform Resource Locator) is used to uniquely identify a resource over the web. URL has the following syntax:

```
protocol://hostname:port/path-and-file-name
```

There are 4 parts in a URL:

1. *Protocol*: The application-level protocol used by the client and server, e.g., HTTP, FTP, and telnet.
2. *Hostname*: The DNS domain name (e.g., `www.test101.com`) or IP address (e.g., 192.128.1.2) of the server.
3. *Port*: The TCP port number that the server is listening for incoming requests from the clients.
4. *Path-and-file-name*: The name and location of the requested resource, under the server document base directory.

For example, in the URL `http://www.test101.com/docs/index.html`, the communication protocol is HTTP; the hostname is `www.test101.com`. The port number was not specified in the URL, and takes on the default number, which is TCP port 80 for HTTP. The path and file name for the resource to be located is "`/docs/index.html`".

Other examples of URL are:

```
ftp://www.ftp.org/docs/test.txt
mailto:user@test101.com
news:soc.culture.Singapore
telnet://www.test101.com/
```

HTTP Protocol

As mentioned, whenever you enter a URL in the address box of the browser, the browser translates the URL into a request message according to the specified protocol; and sends the request message to the server.

For example, the browser translated the URL `http://www.test101.com/doc/index.html` into the following request message:

```
GET /docs/index.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
(blank line)
```

When this request message reaches the server, the server can take either one of these actions:

1. The server interprets the request received, maps the request into a *file* under the server's document directory, and returns the file requested to the client.
2. The server interprets the request received, maps the request into a *program* kept in the server, executes the program, and returns the output of the program to the client.
3. The request cannot be satisfied, the server returns an error message.

An example of the HTTP response message is as shown:

```
HTTP/1.1 200 OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

<html><body><h1>It works!</h1></body></html>
```

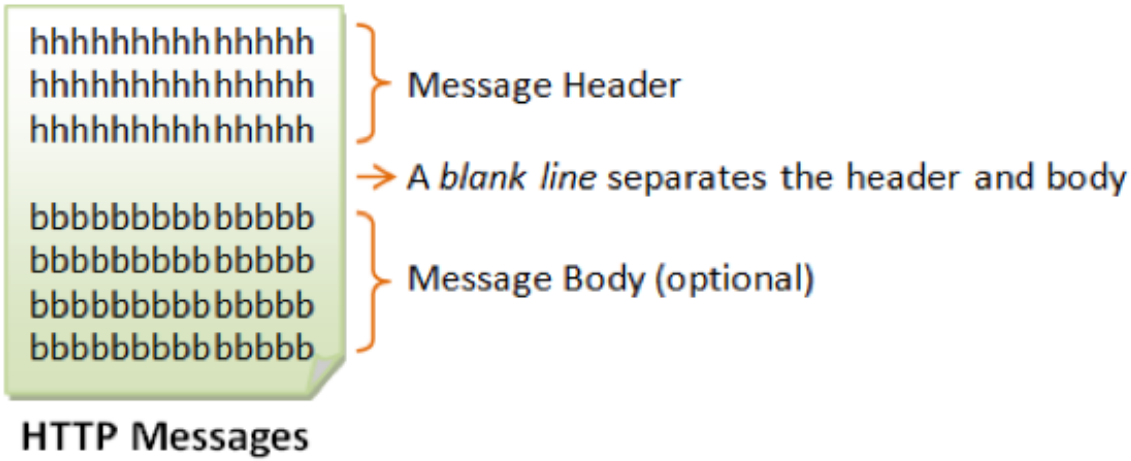
The browser receives the response message, interprets the message and displays the contents of the message on the browser's window according to the media type of the response (as in the Content-Type response header). Common media type include "`text/plain`", "`text/html`", "`image/gif`", "`image/jpeg`", "`audio/mpeg`", "`video/mpeg`", "`application/msword`", and "`application/pdf`".

In its idling state, an HTTP server does nothing but listening to the IP address(es) and port(s) specified in the configuration for incoming request. When a request arrives, the server analyzes the message header, applies rules specified in the configuration, and takes the appropriate action. The webmaster's main control over the action of web server is via the configuration, which will be dealt with in greater details in the later sections.

HTTP Request and Response Messages

HTTP client and server communicate by sending text messages. The client sends a *request message* to the server. The server, in turn, returns a *response message*.

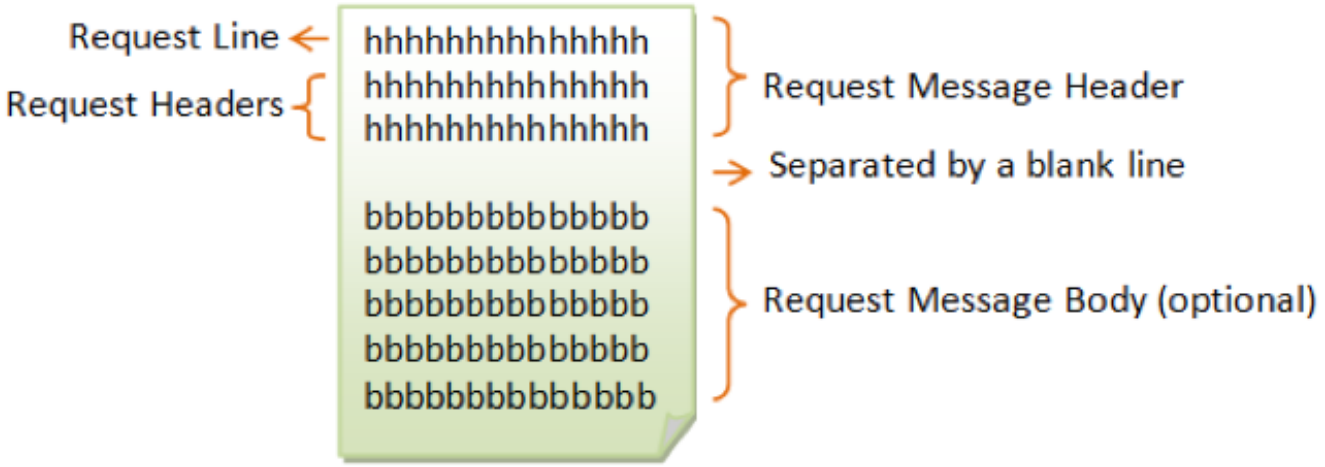
An HTTP message consists of a *message header* and an optional *message body*, separated by a *blank line*, as illustrated below:



HTTP Messages

HTTP Request Message

The format of an HTTP request message is as follow:



HTTP Request Message

Request Line

The first line of the header is called the *request line*, followed by optional *request headers*.

The request line has the following syntax:

```
request-method-name request-URI HTTP-version
```

- *request-method-name*: HTTP protocol defines a set of request methods, e.g., GET, POST, HEAD, and OPTIONS. The client can use one of these methods to send a request to the server.
- *request-URI*: specifies the resource requested.
- *HTTP-version*: Two versions are currently in use: HTTP/1.0 and HTTP/1.1.

Examples of request line are:

```
GET /test.html HTTP/1.1
HEAD /query.html HTTP/1.0
POST /index.html HTTP/1.1
```

Request Headers

The request headers are in the form of `name:value` pairs. Multiple values, separated by commas, can be specified.

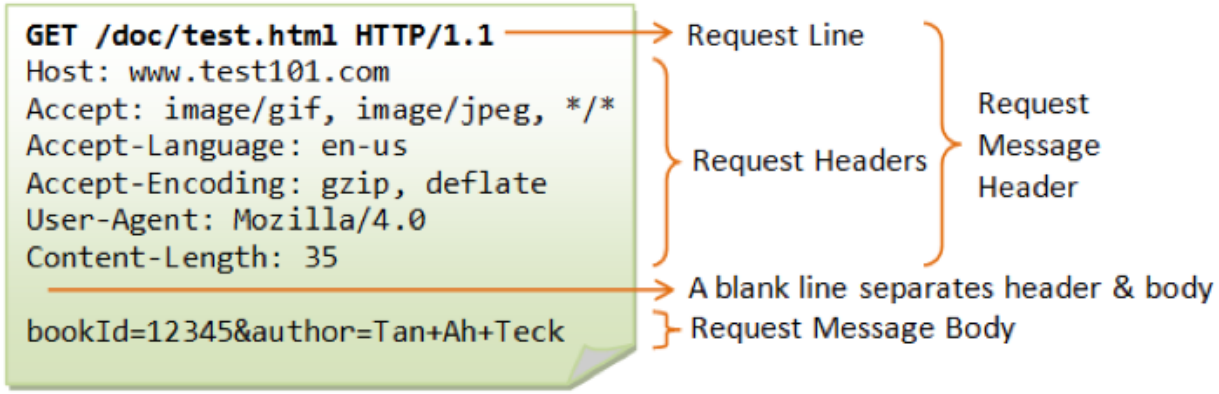
```
request-header-name: request-header-value1, request-header-value2, ...
```

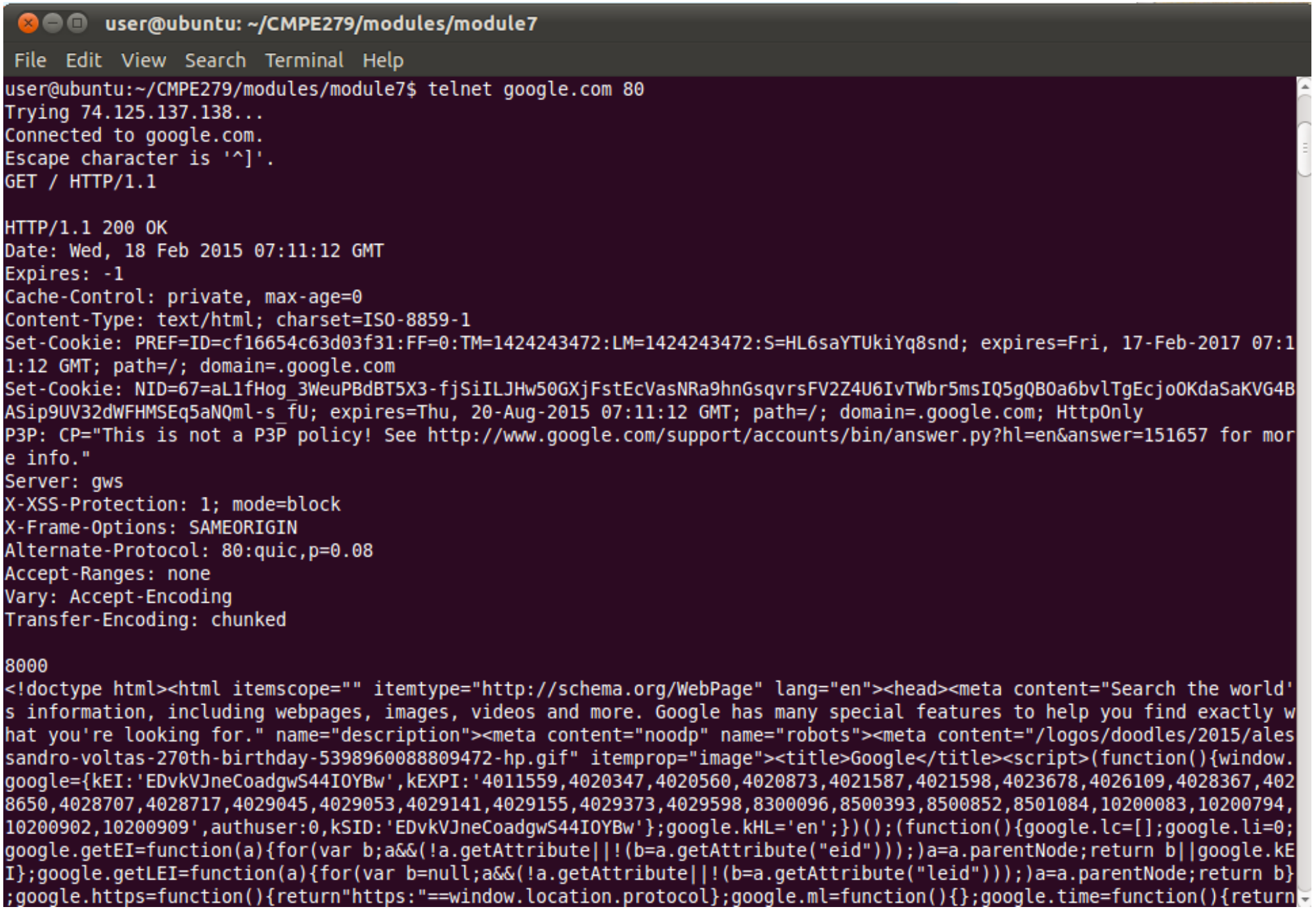
Examples of request headers are:

```
Host: www.xyz.com
Connection: Keep-Alive
Accept: image/gif, image/jpeg, */*
Accept-Language: us-en, fr, cn
```

Example

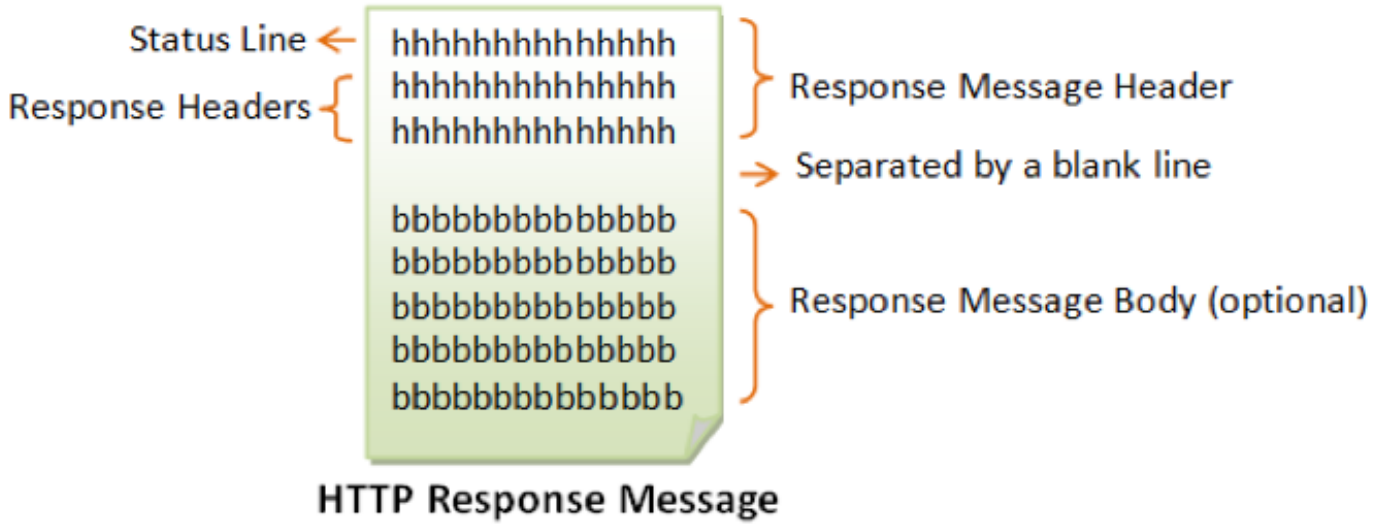
The following shows a sample HTTP request message:




user@ubuntu: ~/CMPE279/modules/module7
File Edit View Search Terminal Help
user@ubuntu:~/CMPE279/modules/module7$ telnet google.com 80
Trying 74.125.137.138...
Connected to google.com.
Escape character is '^]'.
GET / HTTP/1.1
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2015 07:11:12 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Set-Cookie: PREF=ID=cf16654c63d03f31:FF=0:TM=1424243472:LM=1424243472:S=HL6saYTUkiYq8snd; expires=Fri, 17-Feb-2017 07:1
1:12 GMT; path=/; domain=.google.com
Set-Cookie: NID=67=aL1fHog_3WeuPBdBT5X3-fjSiILJHw50GXjFstEcVasNRa9hnGsqvrsFV2Z4U6IvTWbr5msIQ5gQBOa6bvlTgEcjoOKdaSaKVG4B
ASip9UV32dWFHMSEq5aNQml-s_fU; expires=Thu, 20-Aug-2015 07:11:12 GMT; path=/; domain=.google.com; HttpOnly
P3P: CP="This is not a P3P policy! See http://www.google.com/support/accounts/bin/answer.py?hl=en&answer=151657 for mor
e info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Alternate-Protocol: 80:quic,p=0.08
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked
8000
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head><meta content="Search the world'
s information, including webpages, images, videos and more. Google has many special features to help you find exactly w
hat you're looking for." name="description"><meta content="noodp" name="robots"><meta content="/logos/doodles/2015/ales
sandro-voltas-270th-birthday-5398960088809472-hp.gif" itemprop="image"><title>Google</title><script>(function(){window.
google={kEI:'EDvkVJneCoadgwS44IOYBw',kEXPI:'4011559,4020347,4020560,4020873,4021587,4021598,4023678,4026109,4028367,402
8650,4028707,4028717,4029045,4029053,4029141,4029155,4029373,4029598,8300096,8500393,8500852,8501084,10200083,10200794,
10200902,10200909',authuser:0,kSID:'EDvkVJneCoadgwS44IOYBw'};google.kHL='en';})();(function(){google.lc=[];google.li=0;
google.getEI=function(a){for(var b;a&&(!a.getAttribute||!(b=a.getAttribute("eid")));)a=a.parentNode;return b||google.kE
I};google.getLEI=function(a){for(var b=null;a&&(!a.getAttribute||!(b=a.getAttribute("leid")));)a=a.parentNode;return b}
;google.https=function(){return"https:"==window.location.protocol};google.ml=function(){};google.time=function(){return

HTTP Response Message

The format of the HTTP response message is as follows:



HTTP Response Message

Status Line

The first line is called the *status line*, followed by optional response header(s).

The status line has the following syntax:

```
HTTP-version status-code reason-phrase
```

- *HTTP-version*: The HTTP version used in this session. Either HTTP/1.0 and HTTP/1.1.
- *status-code*: a 3-digit number generated by the server to reflect the outcome of the request.
- *reason-phrase*: gives a short explanation to the status code.
- Common status code and reason phrase are "200 OK", "404 Not Found", "403 Forbidden", "500 Internal Server Error".

Examples of status line are:

```
HTTP/1.1 200 OK
HTTP/1.0 404 Not Found
HTTP/1.1 403 Forbidden
```

Response Headers

The response headers are in the form `name:value` pairs:

```
response-header-name: response-header-value1, response-header-value2, ...
```

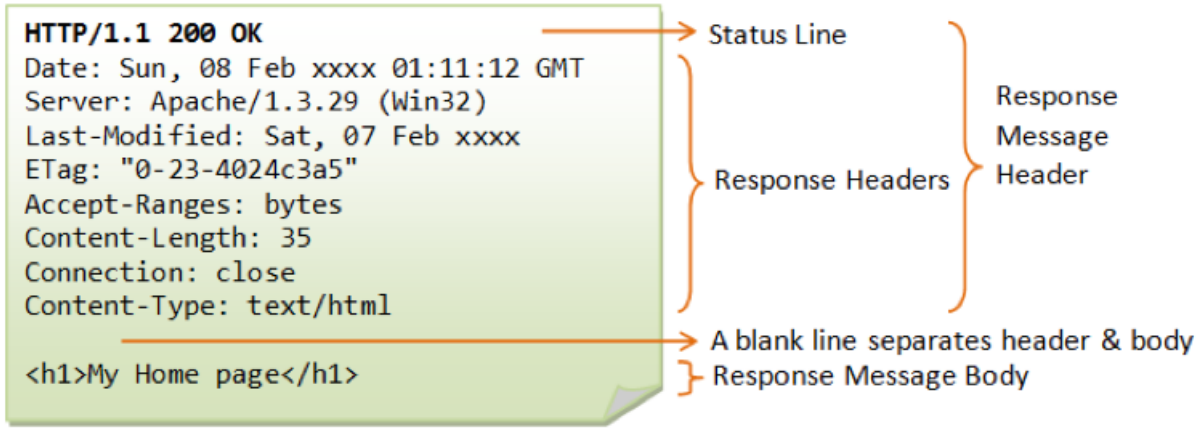
Examples of response headers are:

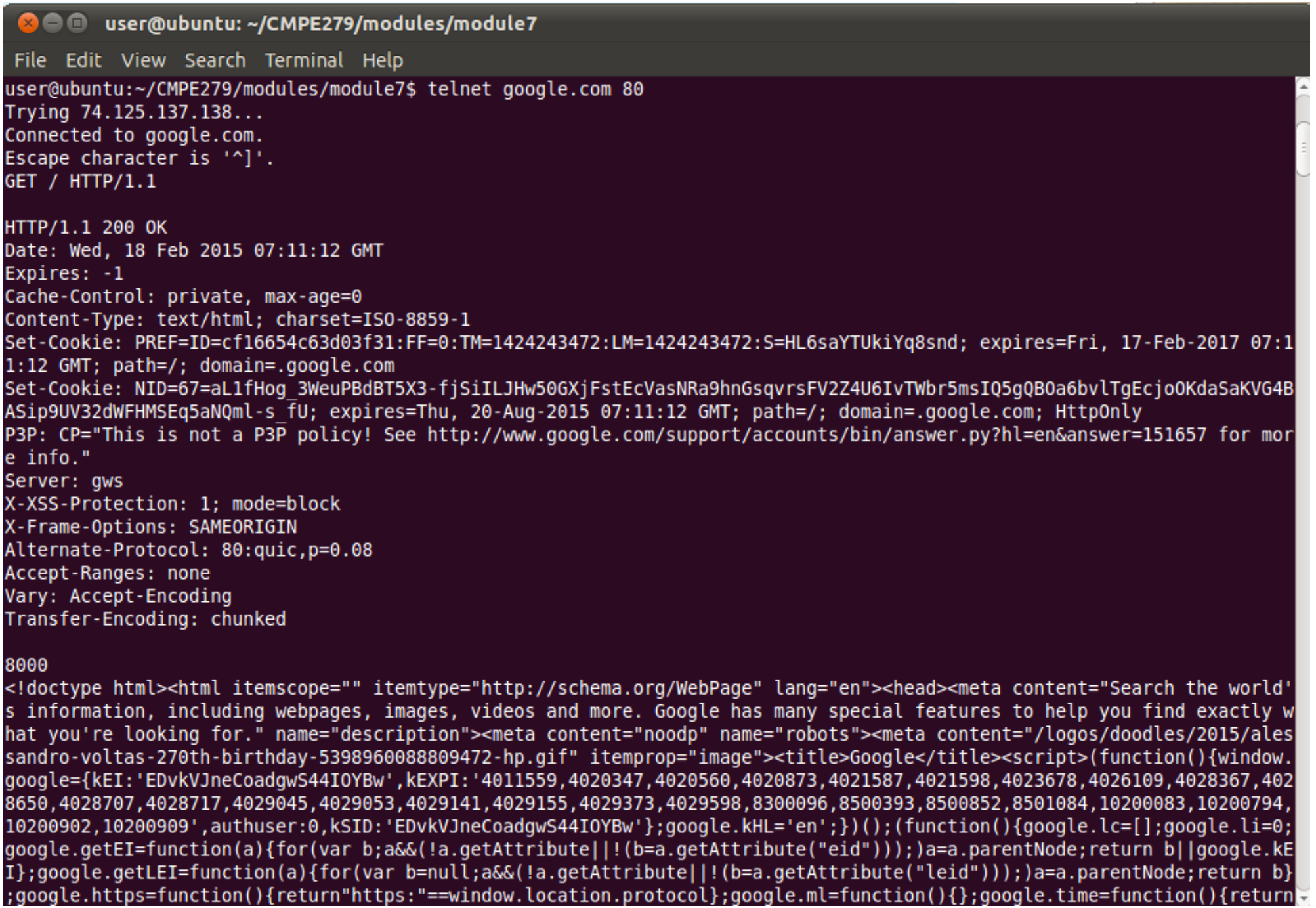
```
Content-Type: text/html
Content-Length: 35
Connection: Keep-Alive
Keep-Alive: timeout=15, max=100
```

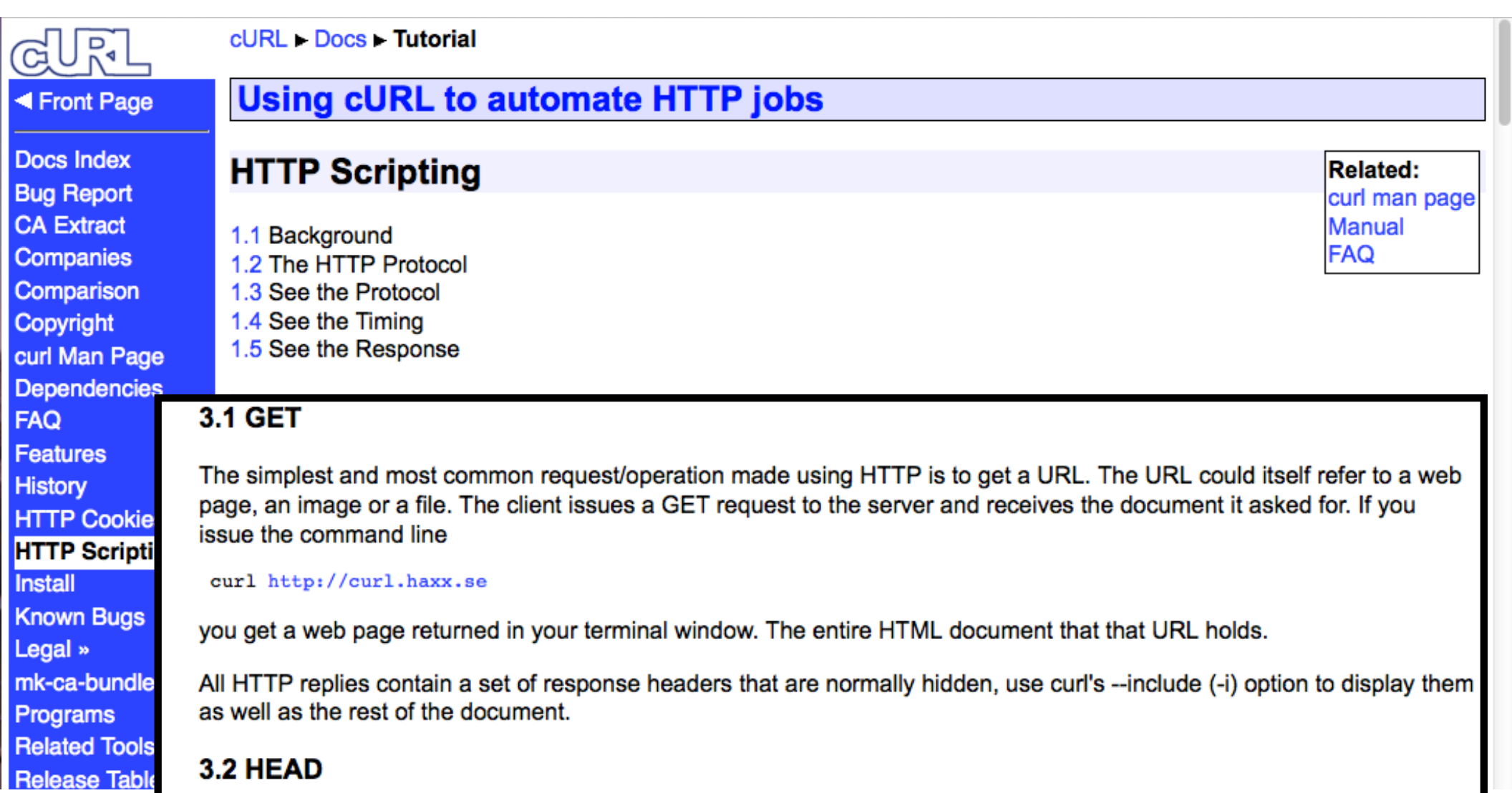
The response message body contains the resource data requested.

Example

The following shows a sample response message:




user@ubuntu: ~/CMPE279/modules/module7
File Edit View Search Terminal Help
user@ubuntu:~/CMPE279/modules/module7$ telnet google.com 80
Trying 74.125.137.138...
Connected to google.com.
Escape character is '^]'.
GET / HTTP/1.1
HTTP/1.1 200 OK
Date: Wed, 18 Feb 2015 07:11:12 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Set-Cookie: PREF=ID=cf16654c63d03f31:FF=0:TM=1424243472:LM=1424243472:S=HL6saYTUkiYq8snd; expires=Fri, 17-Feb-2017 07:1
1:12 GMT; path=/; domain=.google.com
Set-Cookie: NID=67=aL1fHog_3WeuPBdBT5X3-fjSiILJHw50GXjFstEcVasNRa9hnGsqvrsFV2Z4U6IvTWbr5msIQ5gQBOa6bvlTgEcjoOKdaSaKVG4B
ASip9UV32dWFHMSEq5aNQml-s_fU; expires=Thu, 20-Aug-2015 07:11:12 GMT; path=/; domain=.google.com; HttpOnly
P3P: CP="This is not a P3P policy! See http://www.google.com/support/accounts/bin/answer.py?hl=en&answer=151657 for mor
e info."
Server: gws
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Alternate-Protocol: 80:quic,p=0.08
Accept-Ranges: none
Vary: Accept-Encoding
Transfer-Encoding: chunked
8000
<!doctype html><html itemscope="" itemtype="http://schema.org/WebPage" lang="en"><head><meta content="Search the world'
s information, including webpages, images, videos and more. Google has many special features to help you find exactly w
hat you're looking for." name="description"><meta content="noodp" name="robots"><meta content="/logos/doodles/2015/ales
sandro-voltas-270th-birthday-5398960088809472-hp.gif" itemprop="image"><title>Google</title><script>(function(){window.
google={kEI:'EDvkVJneCoadgwS44IOYBw',kEXPI:'4011559,4020347,4020560,4020873,4021587,4021598,4023678,4026109,4028367,402
8650,4028707,4028717,4029045,4029053,4029141,4029155,4029373,4029598,8300096,8500393,8500852,8501084,10200083,10200794,
10200902,10200909',authuser:0,kSID:'EDvkVJneCoadgwS44IOYBw'};google.kHL='en';})();(function(){google.lc=[];google.li=0;
google.getEI=function(a){for(var b;a&&(!a.getAttribute||!(b=a.getAttribute("eid")));)a=a.parentNode;return b||google.kE
I};google.getLEI=function(a){for(var b=null;a&&(!a.getAttribute||!(b=a.getAttribute("leid")));)a=a.parentNode;return b}
;google.https=function(){return"https:"==window.location.protocol};google.ml=function(){};google.time=function(){return



cURL ▸ Docs ▸ Tutorial

◀ Front Page

Docs Index
Bug Report
CA Extract
Companies
Comparison
Copyright
curl Man Page
Dependencies
FAQ
Features
History
HTTP Cookie
HTTP Scripti
Install
Known Bugs
Legal »
mk-ca-bundle
Programs
Related Tools
Release Table

Using cURL to automate HTTP jobs

HTTP Scripting

Related:
curl man page
Manual
FAQ

3.1 GET

The simplest and most common request/operation made using HTTP is to get a URL. The URL could itself refer to a web page, an image or a file. The client issues a GET request to the server and receives the document it asked for. If you issue the command line

```
curl http://curl.haxx.se
```

you get a web page returned in your terminal window. The entire HTML document that that URL holds.

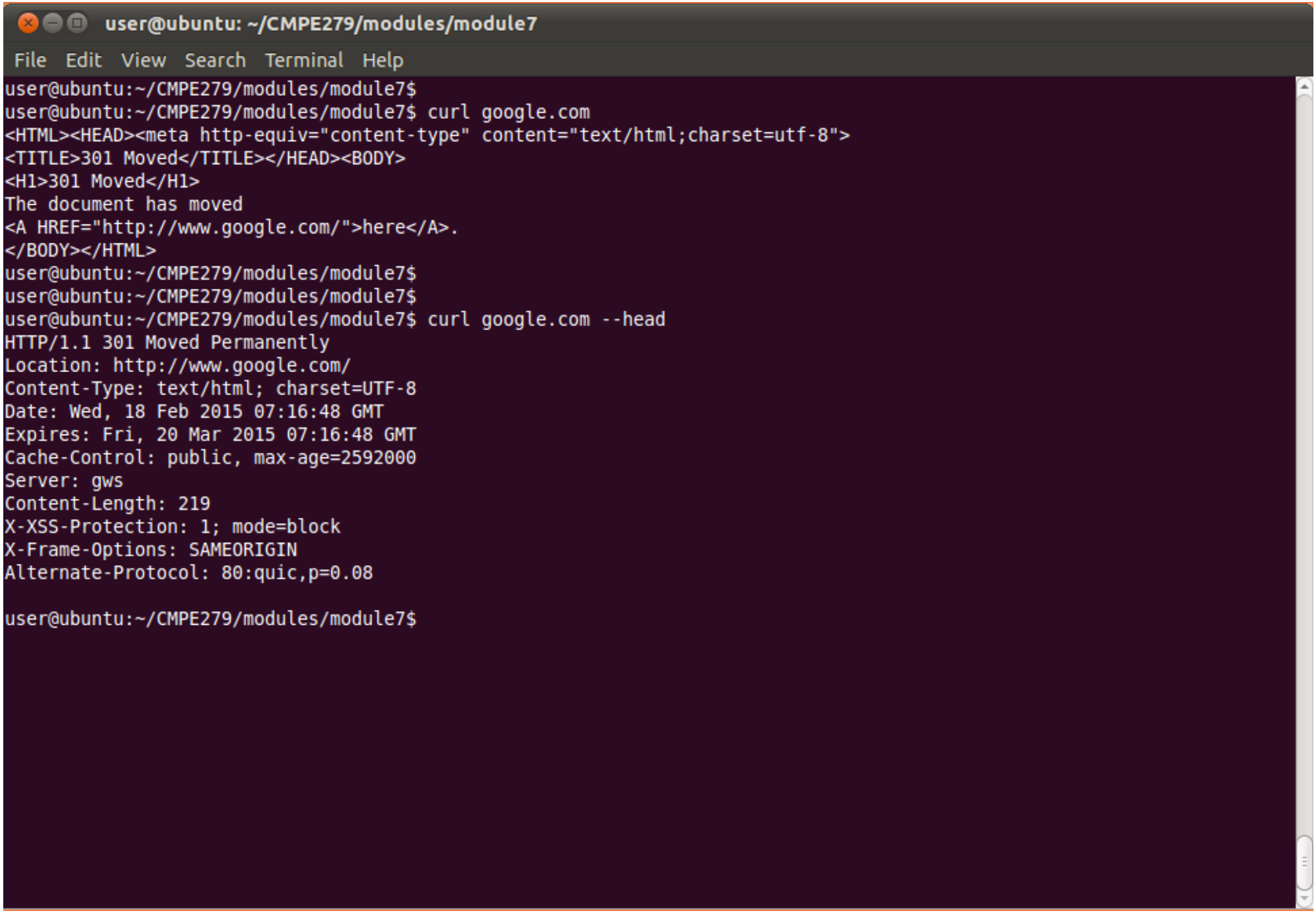
All HTTP replies contain a set of response headers that are normally hidden, use curl's --include (-i) option to display them as well as the rest of the document.

3.2 HEAD

You can ask the remote server for ONLY the headers by using the --head (-I) option which will make curl issue a HEAD request. In some special cases servers deny the HEAD method while others still work, which is a particular kind of annoyance.

The HEAD method is defined and made so that the server returns the headers exactly the way it would do for a GET, but without a body. It means that you may see a Content-Length: in the response headers, but there must not be an actual body in the HEAD response.

http://curl.haxx.se/docs/httpscripting.html


user@ubuntu: ~/CMPE279/modules/module7
File Edit View Search Terminal Help
user@ubuntu:~/CMPE279/modules/module7$
user@ubuntu:~/CMPE279/modules/module7$ curl google.com
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.com/">here</A>.
</BODY></HTML>
user@ubuntu:~/CMPE279/modules/module7$
user@ubuntu:~/CMPE279/modules/module7$
user@ubuntu:~/CMPE279/modules/module7$ curl google.com --head
HTTP/1.1 301 Moved Permanently
Location: http://www.google.com/
Content-Type: text/html; charset=UTF-8
Date: Wed, 18 Feb 2015 07:16:48 GMT
Expires: Fri, 20 Mar 2015 07:16:48 GMT
Cache-Control: public, max-age=2592000
Server: gws
Content-Length: 219
X-XSS-Protection: 1; mode=block
X-Frame-Options: SAMEORIGIN
Alternate-Protocol: 80:quic,p=0.08
user@ubuntu:~/CMPE279/modules/module7$

HTTP Request Methods

HTTP protocol defines a set of request methods. A client can use one of these request methods to send a request message to an HTTP server. The methods are:

- GET: A client can use the GET request to get a web resource from the server.
- HEAD: A client can use the HEAD request to get the header that a GET request would have obtained. Since the header contains the last-modified date of the data, this can be used to check against the local cache copy.
- POST: Used to post data up to the web server.
- PUT: Ask the server to store the data.
- DELETE: Ask the server to delete the data.
- TRACE: Ask the server to return a diagnostic trace of the actions it takes.
- OPTIONS: Ask the server to return the list of request methods it supports.
- CONNECT: Used to tell a proxy to make a connection to another host and simply reply the content, without attempting to parse or cache it. This is often used to make SSL connection through the proxy.
- Other extension methods.

POST, GET, PUT, DELETE
are also known as
CRUD Operations (respectively)

Response Status Code

The first line of the response message (i.e., the status line) contains the response status code, which is generated by the server to indicate the outcome of the request.

The status code is a 3-digit number:

- 1xx (Informational): Request received, server is continuing the process.
- 2xx (Success): The request was successfully received, understood, accepted and serviced.
- 3xx (Redirection): Further action must be taken in order to complete the request.
- 4xx (Client Error): The request contains bad syntax or cannot be understood.
- 5xx (Server Error): The server failed to fulfill an apparently valid request.

Some commonly encountered status codes are:

- 100 Continue: The server received the request and in the process of giving the response.
- 200 OK: The request is fulfilled.
- 301 Move Permanently: The resource requested for has been permanently moved to a new location. The URL of the new location is given in the response header called `Location`. The client should issue a new request to the new location. Application should update all references to this new location.
- 302 Found & Redirect (or Move Temporarily): Same as 301, but the new location is temporarily in nature. The client should issue a new request, but applications need not update the references.
- 304 Not Modified: In response to the `If-Modified-Since` conditional GET request, the server notifies that the resource requested has not been modified.
- 400 Bad Request: Server could not interpret or understand the request, probably syntax error in the request message.
- 401 Authentication Required: The requested resource is protected, and require client's credential (username/password). The client should re-submit the request with his credential (username/password).
- 403 Forbidden: Server refuses to supply the resource, regardless of identity of client.
- 404 Not Found: The requested resource cannot be found in the server.
- 405 Method Not Allowed: The request method used, e.g., POST, PUT, DELETE, is a valid method. However, the server does not allow that method for the resource requested.
- 408 Request Timeout:
- 414 Request URI too Large:
- 500 Internal Server Error: Server is confused, often caused by an error in the server-side program responding to the request.
- 501 Method Not Implemented: The request method used is invalid (could be caused by a typing error, e.g., "GET" misspell as "Get").
- 502 Bad Gateway: Proxy or Gateway indicates that it receives a bad response from the upstream server.
- 503 Service Unavailable: Server cannot response due to overloading or maintenance. The client can try again later.
- 504 Gateway Timeout: Proxy or Gateway indicates that it receives a timeout from an upstream server.

More HTTP/1.0 GET Request Examples

Example: Misspelt Request Method

In the request, "GET" is misspelled as "get". The server returns an error "501 Method Not Implemented". The response header "Allow" tells the client the methods allowed.

```
get /test.html HTTP/1.0
(enter twice to create a blank line)
```

```
HTTP/1.1 501 Method Not Implemented
Date: Sun, 18 Oct 2009 10:32:05 GMT
Server: Apache/2.2.14 (Win32)
Allow: GET,HEAD,POST,OPTIONS,TRACE
Content-Length: 215
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>501 Method Not Implemented</title>
</head><body>
<h1>Method Not Implemented</h1>
<p>get to /index.html not supported.<br />
</p>
</body></html>
```

Example: 404 File Not Found

In this GET request, the *request-URL* "`/t.html`" cannot be found under the server's document directory. The server returns an error "404 Not Found".

```
GET /t.html HTTP/1.0
(enter twice to create a blank line)
```

```
HTTP/1.1 404 Not Found
Date: Sun, 18 Oct 2009 10:36:20 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 204
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL /t.html was not found on this server.</p>
</body></html>
```

Example: Wrong HTTP Version Number

In this GET request, the HTTP-version was misspelled, resulted in bad syntax. The server returns an error "404 Bad Request". HTTP-version should be either HTTP/1.0 or HTTP/1.1.

```
GET /index.html HTTTTTP/1.0
(enter twice to create a blank line)
```

```
HTTP/1.1 400 Bad Request
Date: Sun, 08 Feb 2004 01:29:40 GMT
Server: Apache/1.3.29 (Win32)
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<HTML><HEAD>
<TITLE>400 Bad Request</TITLE>
</HEAD><BODY>
<H1>Bad Request</H1>
Your browser sent a request that this server could not understand.<P>
The request line contained invalid characters following the protocol string.<P><P>
</BODY></HTML>
```

Note: The latest Apache 2.2.14 ignores this error and returns the document with status code "200 OK".

Example: Wrong Request-URI

In the following GET request, the *request-URI* did not begin from the root "/", resulted in a "bad request".

```
GET test.html HTTP/1.0
(blank line)
```

```
HTTP/1.1 400 Bad Request
Date: Sun, 18 Oct 2009 10:42:27 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 226
Connection: close
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>400 Bad Request</title>
</head><body>
<h1>Bad Request</h1>
<p>Your browser sent a request that this server could not understand.<br />
</p>
</body></html>
```

Example: Accessing a Protected Resource

The following GET request tried to access a protected resource. The server returns an error "403 Forbidden". In this example, the directory "`htdocs\forbidden`" is configured to deny all access in the Apache HTTP server configuration file "`httpd.conf`" as follows:

```
<Directory "C:/apache/htdocs/forbidden">
   Order deny,allow
   deny from all
</Directory>
```

```
GET /forbidden/index.html HTTP/1.0
(blank line)
```

```
HTTP/1.1 403 Forbidden
Date: Sun, 18 Oct 2009 11:58:41 GMT
Server: Apache/2.2.14 (Win32)
Content-Length: 222
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access /forbidden/index.html
on this server.</p>
</body></html>
```

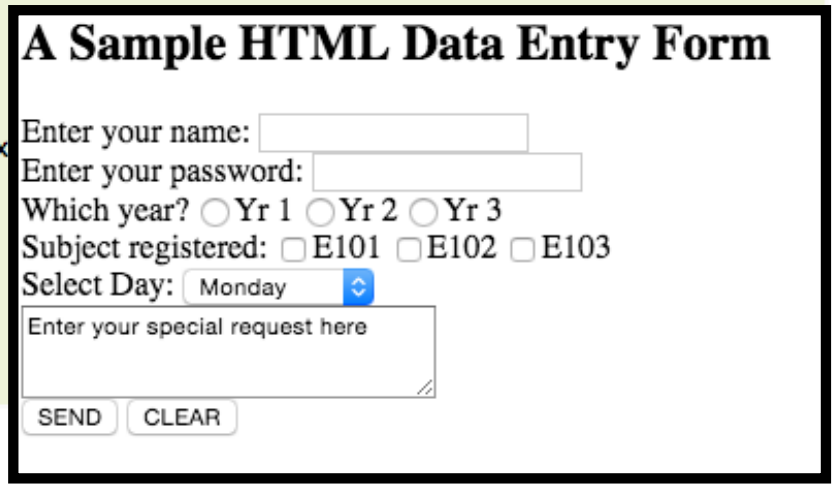
Submitting HTML Form Data and Query String

In many Internet applications, such as e-commerce and search engine, the clients are required to submit additional information to the server (e.g., the name, address, the search keywords). Based on the data submitted, the server takes an appropriate action and produces a customized response.

The clients are usually presented with a form (produced using HTML <form> tag). Once they fill in the requested data and hit the submit button, the browser packs the form data and submits them to the server, using either a GET request or a POST request.

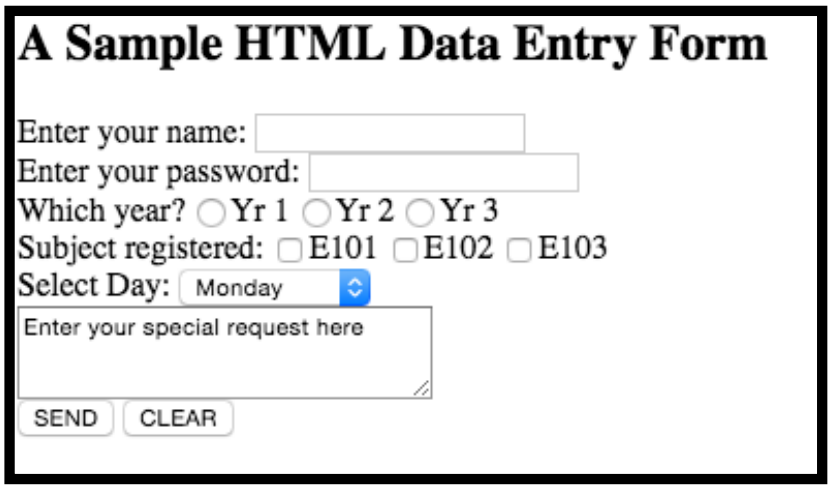
The following is a sample HTML form, which is produced by the following HTML script:

```
<html>
<head><title>A Sample HTML Form</title></head>
<body>
  <h2 align="left">A Sample HTML Data Entry Form</h2>
  <form method="get" action="/bin/process">
    Enter your name: <input type="text" name="username"><br />
    Enter your password: <input type="password" name="password"><br />
    Which year?
    <input type="radio" name="year" value="2" />Yr 1
    <input type="radio" name="year" value="2" />Yr 2
    <input type="radio" name="year" value="3" />Yr 3<br />
    Subject registered:
    <input type="checkbox" name="subject" value="e101" />E101
    <input type="checkbox" name="subject" value="e102" />E102
    <input type="checkbox" name="subject" value="e103" />E103<br />
    Select Day:
    <select name="day">
      <option value="mon">Monday</option>
      <option value="wed">Wednesday</option>
      <option value="fri">Friday</option>
    </select><br />
    <textarea rows="3" cols="30">Enter your special request here</tex
    <input type="submit" value="SEND" />
    <input type="reset" value="CLEAR" />
    <input type="hidden" name="action" value="registration" />
  </form>
</body>
</html>
```



A form contains fields. The types of field include:

- Text Box: produced by `<input type="text">`.
- Password Box: produced by `<input type="password">`.
- Radio Button: produced by `<input type="radio">`.
- Checkbox: produced by `<input type="checkbox">`.
- Selection: produced by `<select>` and `<option>`.
- Text Area: produced by `<textarea>`.
- Submit Button: produced by `<input type="submit">`.
- Reset Button: produced by `<input type="reset">`.
- Hidden Field: produced by `<input type="hidden">`.
- Button: produced by `<input type="button">`.

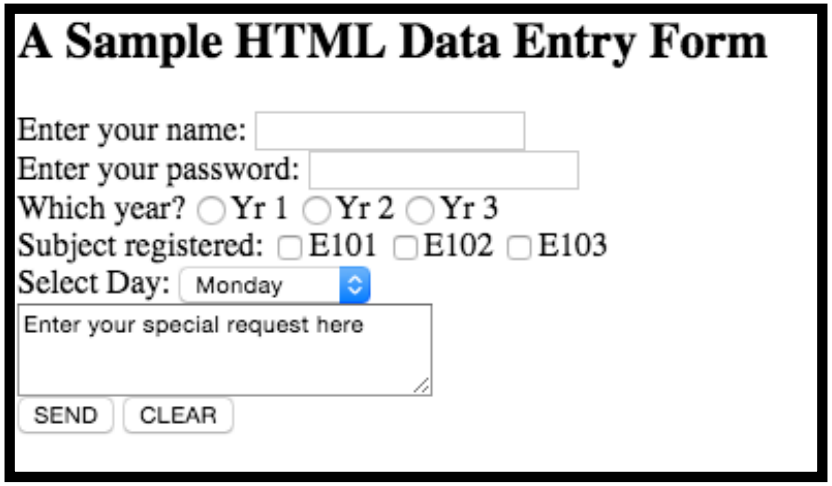


Each field has a *name* and can take on a specified *value*. Once the client fills in the fields and hits the submit button, the browser gathers each of the fields' name and value, packed them into "`name=value`" pairs, and concatenates all the fields together using "`&`" as the field separator. This is known as a *query string*. It will send the query string to the server as part of the request.

```
name1=value1&name2=value2&name3=value3&...
```

Special characters are not allowed inside the query string. They must be replaced by a "`%`" followed by the ASCII code in Hex. E.g., "`~`" is replaced by "`%7E`", "`#`" by "`%23`" and so on. Since blank is rather common, it can be replaced by either "`%20`" or "`+`" (the "`+`" character must be replaced by "`%2B`"). This replacement process is called *URL-encoding*, and the result is a *URL-encoded query string*. For example, suppose that there are 3 fields inside a form, with name/value of "name=Peter Lee", "address=#123 Happy Ave" and "language=C++", the URL-encoded query string is:

```
name=Peter+Lee&address=%23123+Happy+Ave&Language=C%2B%2B
```



The query string can be sent to the server using either HTTP GET or POST request method, which is specified in the `<form>`'s attribute "`method`".

```
<form method="get|post" action="url">
```

If GET request method is used, the URL-encoded query string will be *appended* behind the *request-URI* after a "?" character, i.e.,

```
GET request-URI?query-string HTTP-version
(other optional request headers)
(blank line)
(optional request body)
```

Using GET request to send the query string has the following drawbacks:

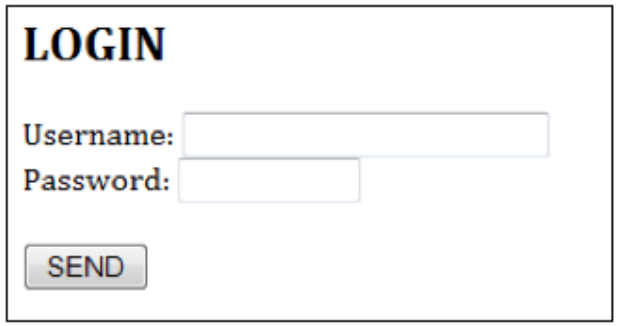
- The amount of data you could append behind *request-URI* is limited. If this amount exceed a server-specific threshold, the server would return an error "414 Request URI too Large".
- The URL-encoded query string would appear on the address box of the browser.

POST method overcomes these drawbacks. If POST request method is used, the query string will be sent in the body of the request message, where the amount is not limited. The request headers `Content-Type` and `Content-Length` are used to notify the server the type and the length of the query string. The query string will not appear on the browser's address box. POST method will be discussed later.

Example

The following HTML form is used to gather the username and password in a login menu.

```
<html>
<head><title>Login</title></head>
<body>
  <h2>LOGIN</h2>
  <form method="get" action="/bin/login">
    Username: <input type="text" name="user" size="25" /><br />
    Password: <input type="password" name="pw" size="10" /><br /><br />
    <input type="hidden" name="action" value="login" />
    <input type="submit" value="SEND" />
  </form>
</body>
</html>
```

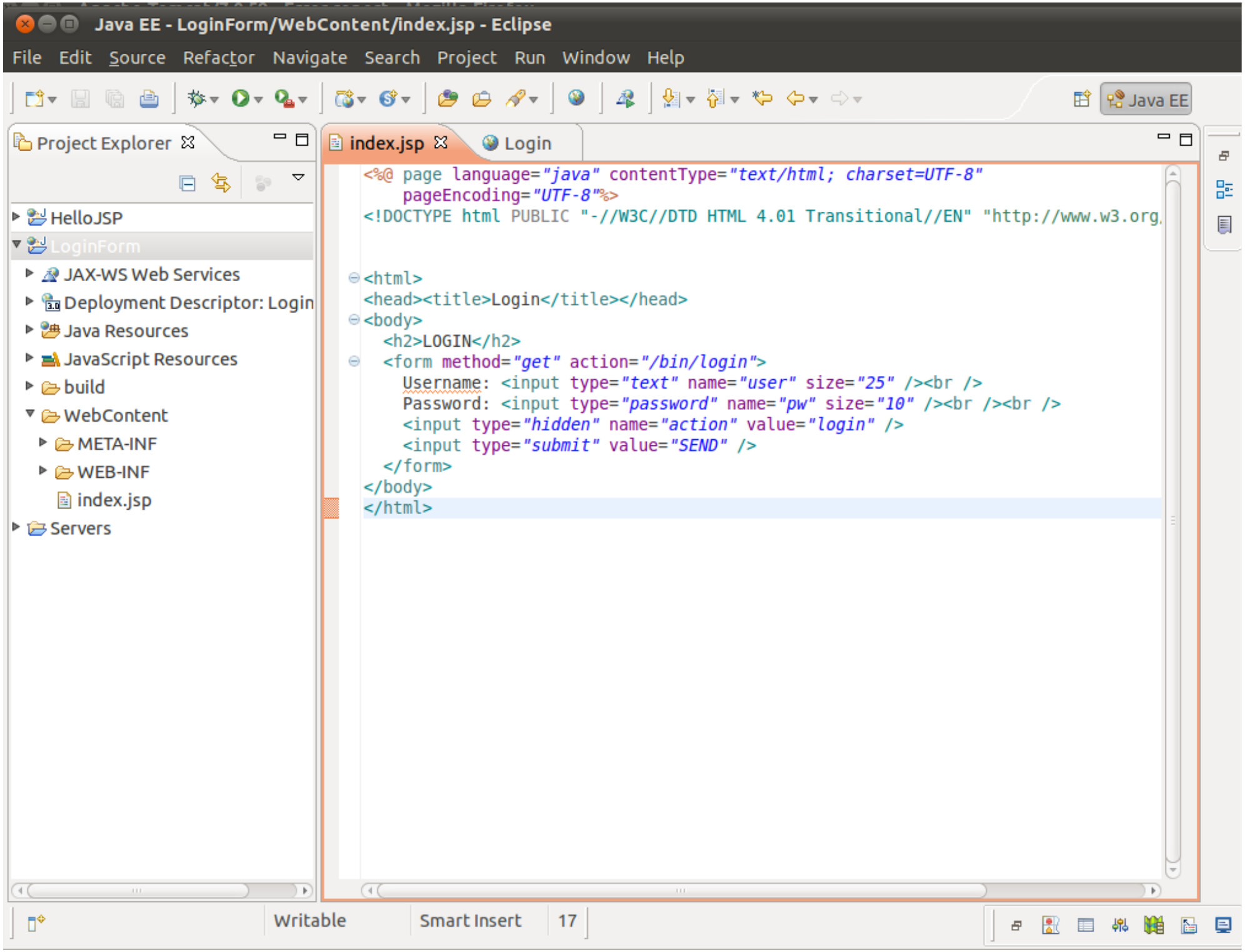


The HTTP GET request method is used to send the query string. Suppose the user enters "Peter Lee" as the username, "123456" as password; and clicks the submit button. The following GET request is:

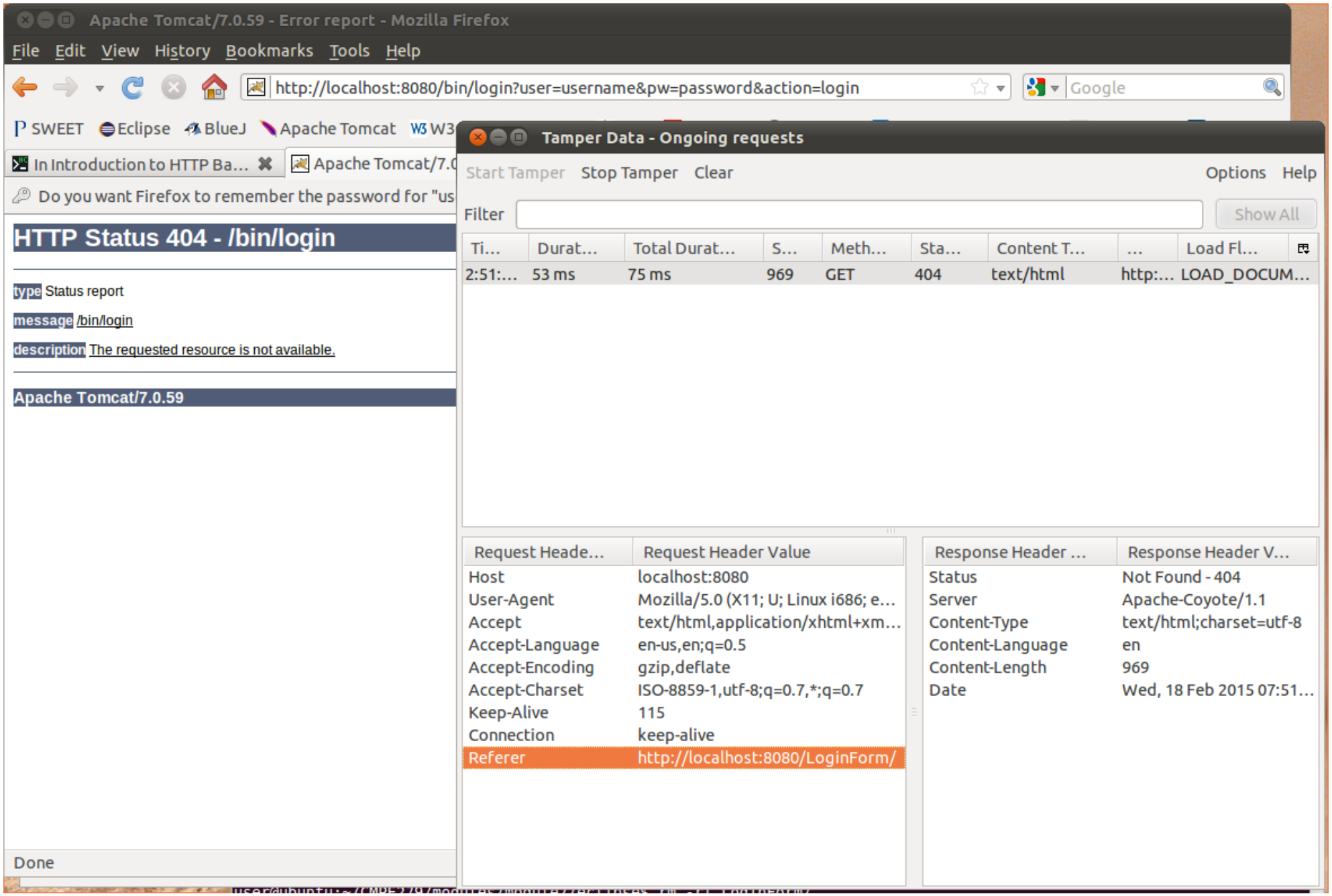
```
GET /bin/login?user=Peter+Lee&pw=123456&action=login HTTP/1.1
Accept: image/gif, image/jpeg, */*
Referer: http://127.0.0.1:8000/login.html
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: 127.0.0.1:8000
Connection: Keep-Alive
```

Note that although the password that you enter does not show on the screen, it is shown clearly in the address box of the browser. You should never use send your password without proper encryption.

```
http://127.0.0.1:8000/bin/login?user=Peter+Lee&pw=123456&action=login
```


Java EE - LoginForm/WebContent/index.jsp - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Java EE
Project Explorer
HelloJSP
LoginForm
JAX-WS Web Services
Deployment Descriptor: Login
Java Resources
JavaScript Resources
build
WebContent
META-INF
WEB-INF
index.jsp
Servers
index.jsp
Login
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org
<html>
<head><title>Login</title></head>
<body>
<h2>LOGIN</h2>
<form method="get" action="/bin/login">
Username: <input type="text" name="user" size="25" />
Password: <input type="password" name="pw" size="10" />

<input type="hidden" name="action" value="login" />
<input type="submit" value="SEND" />
</form>
</body>
</html>
Writable
Smart Insert
17


Apache Tomcat/7.0.59 - Error report - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/bin/login?user=username&pw=password&action=login
Google
SWEET Eclipse BlueJ Apache Tomcat W3
In Introduction to HTTP Ba...
Apache Tomcat/7.0
Do you want Firefox to remember the password for "us
HTTP Status 404 - /bin/login
type Status report
message /bin/login
description The requested resource is not available.
Apache Tomcat/7.0.59
Done
Tamper Data - Ongoing requests
Start Tamper Stop Tamper Clear
Options Help
Filter
Show All
Ti... Durat... Total Durat... S... Meth... Sta... Content T... ... Load Fl...
2:51:... 53 ms 75 ms 969 GET 404 text/html http:... LOAD_DOCUM...
Request Heade... Request Header Value
Host localhost:8080
User-Agent Mozilla/5.0 (X11; U; Linux i686; e...
Accept text/html,application/xhtml+xm...
Accept-Language en-us,en;q=0.5
Accept-Encoding gzip,deflate
Accept-Charset ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive 115
Connection keep-alive
Referer http://localhost:8080/LoginForm/
Response Header ... Response Header V...
Status Not Found - 404
Server Apache-Coyote/1.1
Content-Type text/html;charset=utf-8
Content-Language en
Content-Length 969
Date Wed, 18 Feb 2015 07:51...

URL and URI

URL (Uniform Resource Locator)

A URL (Uniform Resource Locator), defined in RFC 2396, is used to uniquely identify a resource over the web. URL has the following syntax:

```
protocol://hostname:port/path-and-file-name
```

There are 4 parts in a URL:

1. *Protocol*: The application-layer protocol used by the client and server, e.g., HTTP, FTP, and telnet.
2. *Hostname*: The DNS domain name (e.g., www.test101.com) or IP address (e.g., 192.128.1.2) of the server.
3. *Port*: The TCP port number that the server is listening for incoming requests from the clients.
4. *Path-and-file-name*: The name and location of the requested resource, under the server document base directory.

For example, in the URL `http://www.test101.com/docs/index.html`, the communication protocol is HTTP; the hostname is www.test101.com. The port number was not specified in the URL, and takes on the default number, which is TCP port 80 for HTTP [STD 2]. The path and file name for the resource to be located is "`/docs/index.html`".

Other examples of URL are:

```
ftp://www.ftp.org/docs/test.txt
mailto:user@test101.com
news:soc.culture.Singapore
telnet://www.test101.com/
```

Encoded URL

URL cannot contain special characters, such as blank or '~'. Special characters are encoded, in the form of %xx, where xx is the ASCII hex code. For example, '~' is encoded as %7e; '+' is encoded as %2b. A blank can be encoded as %20 or '+'. The URL after encoding is called *encoded URL*.

URI (Uniform Resource Identifier)

URI (Uniform Resource Identifier), defined in RFC3986, is more general than URL, which can even locate a *fragment* within a resource. The URI syntax for HTTP protocol is:

```
http://host:port/path?request-parameters#nameAnchor
```

- The request parameters, in the form of name=value pairs, are separated from the URL by a '?'. The name=value pairs are separated by a '&'.
- The #nameAnchor identifies a fragment within the HTML document, defined via the anchor tag ....
- URL rewriting for session management, e.g., "`...;sessionID=xxxxxx`".

"POST" Request Method

POST request method is used to "post" additional data up to the server (e.g., submitting HTML form data or uploading a file). Issuing an HTTP URL from the browser always triggers a GET request. To trigger a POST request, you can use an HTML form with attribute `method="post"` or write your own network program. For submitting HTML form data, POST request is the same as the GET request except that the URL-encoded query string is sent in the request body, rather than appended behind the *request-URI*.

The POST request takes the following syntax:

```
POST request-URI HTTP-version
Content-Type: mime-type
Content-Length: number-of-bytes
(other optional request headers)

(URL-encoded query string)
```

Request headers `Content-Type` and `Content-Length` is necessary in the POST request to inform the server the media type and the length of the request body.

POST vs GET for Submitting Form Data

As mentioned in the previous section, POST request has the following advantage compared with the GET request in sending the query string:

- The amount of data that can be posted is unlimited, as they are kept in the request body, which is often sent to the server in a separate data stream.
- The query string is not shown on the address box of the browser.

Note that although the password is not shown on the browser's address box, it is transmitted to the server in clear text, and subjected to network sniffing. Hence, sending password using a POST request is absolutely not secure.

Example: Submitting Form Data using POST Request Method

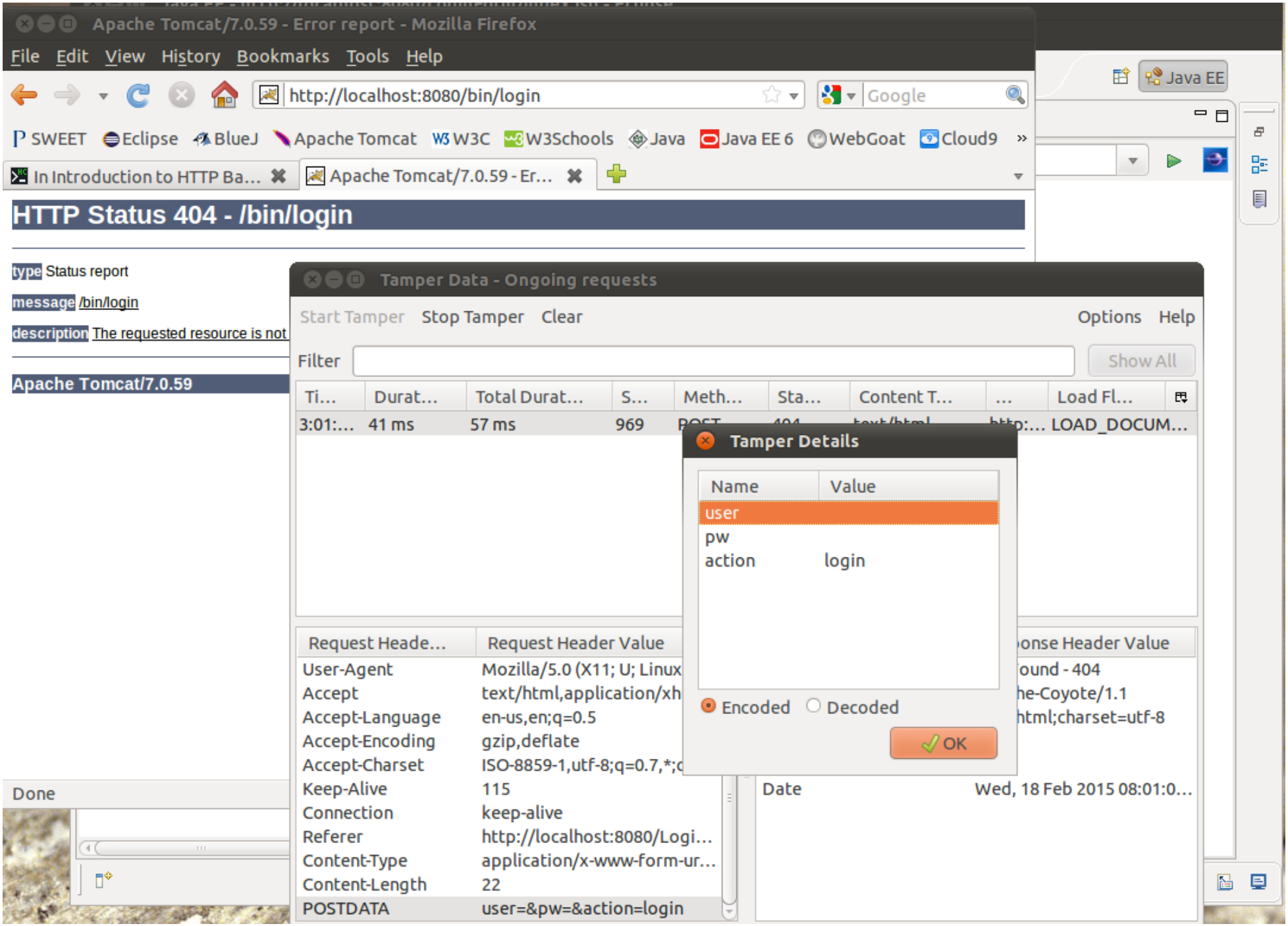
We use the same HTML script as above, but change the request method to POST.

```
<html>
<head><title>Login</title></head>
<body>
  <h2>LOGIN</h2>
  <form method="post" action="/bin/login">
    Username: <input type="text" name="user" size="25" /><br />
    Password: <input type="password" name="pw" size="10" /><br /><br />
    <input type="hidden" name="action" value="login" />
    <input type="submit" value="SEND" />
  </form>
</body>
</html>
```

Suppose the user enters "Peter Lee" as username and "123456" as password, and clicks the submit button, the following POST request would be generated by the browser:

```
POST /bin/login HTTP/1.1
Host: 127.0.0.1:8000
Accept: image/gif, image/jpeg, */*
Referer: http://127.0.0.1:8000/login.html
Accept-Language: en-us
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Content-Length: 37
Connection: Keep-Alive
Cache-Control: no-cache

User=Peter+Lee&pw=123456&action=login
```

Note that the `Content-Type` header informs the server the data is URL-encoded (with a special MIME type `application/x-www-form-urlencoded`), and the `Content-Length` header tells the server how many bytes to read from the message body.


Apache Tomcat/7.0.59 - Error report - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/bin/login
Google
SWEET Eclipse BlueJ Apache Tomcat W3C W3Schools Java Java EE 6 WebGoat Cloud9
In Introduction to HTTP Ba...
Apache Tomcat/7.0.59 - Er...
HTTP Status 404 - /bin/login
type Status report
message /bin/login
description The requested resource is not
Apache Tomcat/7.0.59
Java EE
Tamper Data - Ongoing requests
Start Tamper Stop Tamper Clear
Options Help
Filter
Show All
Ti... Durat... Total Durat... S... Meth... Sta... Content T... ... Load Fl...
3:01:... 41 ms 57 ms 969 LOAD_DOCUM...
Tamper Details
Name Value
user
pw
action login
Encoded Decoded
OK
Request Heade... Request Header Value
User-Agent Mozilla/5.0 (X11; U; Linux
Accept text/html,application/xh
Accept-Language en-us,en;q=0.5
Accept-Encoding gzip,deflate
Accept-Charset ISO-8859-1,utf-8;q=0.7,*;
Keep-Alive 115
Connection keep-alive
Referer http://localhost:8080/Logi...
Content-Type application/x-www-form-ur...
Content-Length 22
POSTDATA user=&pw=&action=login
onse Header Value
ound - 404
he-Coyote/1.1
html;charset=utf-8
Date Wed, 18 Feb 2015 08:01:0...
Done

Cache Control

The client can send a request header "`Cache-control: no-cache`" to tell the proxy to get a fresh copy from the original server, even thought there is a local cached copy. Unfortunately, HTTP/1.0 server does not understand this header, but uses an older request header "`Pragma: no-cache`". You could include both headers in your request.

```
Pragma: no-cache
Cache-Control: no-cache
```

(More, Under Construction)

REFERENCES & RESOURCES

- W3C HTTP specifications at http://www.w3.org/standards/techs/http.
- RFC 2616 "Hypertext Transfer Protocol HTTP/1.1", 1999 @ http://www.ietf.org/rfc/rfc2616.txt.
- RFC 1945 "Hypertext Transfer Protocol HTTP/1.0", 1996 @ http://www.ietf.org/rfc/rfc1945.txt.
- STD 2: "Assigned numbers", 1994.
- STD 5: "Internet Protocol (IP)", 1981.
- STD 6: "User Datagram Protocol (UDP)", 1980.
- STD 7: "Transmission Control Protocol (TCP)", 1983.
- RFC 2396: "Uniform Resource Identifiers (URI): Generic Syntax", 1998.
- RFC 2045: "Multipurpose Internet Mail Extension (MIME) Part 1: Format of Internet Message Bodies", 1996.
- RFC 1867: "Form-based File upload in HTML", 1995, (obsoleted by RFC2854).
- RFC 2854: "The text/html media type", 2000.
- Mutlipart Servlet for file upload @ www.servlets.com

HTML

https://www3.ntu.edu.sg/home/ehchua/programming/index.html

3.1 Basic Layout of an HTML Document

An HTML file begins with a `<!DOCTYPE html>` (document-type) tag to identify itself as an HTML document. The DOCTYPE can be used by the browser to *validate* the HTML file.

An HTML document is contained within a pair of `<html>...</html>` tag. There are two sections in the document: HEAD and BODY, contained within `<head>...</head>` and `<body>...</body>` tags, respectively.

This is a simple example of an HTML document:

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Basic Layout</title>
5  </head>
6  <body>
7    <h2>My First Web Page</h2>
8    <hr />
9    <p>This is my <strong>first</strong> web page written in HTML.</p>
10   <h3>HTML</h3>
11   <p>HTML uses <em>markup tag</em> to <em>markup</em> a document.</p>
12 </body>
13 </html>
```

1. In the HEAD section, the `<title>...</title>` tag provides a *descriptive title* to the page. The browser renders the title on its "title" bar.
2. In the BODY section:
   a. The `<h1>...</h1>` tag marks the enclosing texts as Level-1 Heading. There are six levels of heading in HTML, from `<h1>...</h1>` (largest) to `<h6>...</h6>`. Line 10 uses a `<h3>...</h3>` (heading level-3).
   b. The `<hr />` *standalone* tag (which does not enclose text) draws a horizontal rule (line).
   c. The `<p>...</p>` tag marks the enclosing texts as a paragraph. `<p>` is the most frequently-used tag in HTML.
   d. The nested `<strong>...<strong>` tag (under the `<p>...</p>` tag) specifies "strong emphasis" (to be displayed in bold); and the nested `<em>...</em>` tag specifies "emphasis" (to be displayed in italic).

Use a text editor (I recommend NotePad++) to enter the above HTML codes and save as "`myFirst.html`". Open the file in a browser (such as Firefox, IE, Chrome, Safari, Opera), by double-clicking the file, or drag and drop the file into the browser, or through the browser's "File" menu ⇒ Open File... ⇒ "Browse" and select the file.

Rendering HTML Pages

Browsers follow these rules when rendering HTML documents:

1. HTML is not case sensitive. I recommend using lowercase, as it is easier to type and XHTML compliance. (XML/XHTML is case sensitive.)
2. Blanks, tabs, new-lines and carriage-returns are collectively known as *white spaces*. "Extra" white spaces are ignored. That is, only the first white space is recognized and displayed. For example,

```
<p>See   how the   extra white    spaces,
      tabs  and
    line-breaks    are   ignored  by  the
                browser.</p>
```

produces the following single-line output on screen with words separated by a single space:

```
See how the extra white spaces, tabs and line-breaks are ignored by the browser.
```

You need to use the paragraph tag `<p>...</p>` to layout a paragraph; or insert a manual line-break tag `<br />` to break into a new line. In other words, you control the new-lines via the mark-up tags. To get multiple whitespace, use a special code sequence ` ` (which stands for non-breaking space). For example,

```
<p>This is a
paragraph.</p>
<p>Another paragraph<br />with a line-break in between.</p>
```

This is a paragraph.

Another paragraph
with a line-break in between.

3.2 HTML Tags (aka HTML Elements)

As seen in the above example, HTML tags (aka HTML elements) are enclosed by a pair of angle brackets `<...>`. There are two types of markup tags:

1. **Container Tag:** A container tag has an *opening tag* `<tag-name>` that activates an effect to its content, and has a *matching closing tag* `</tag-name>`, with a leading forward slash, to discontinue the effect. For example:

```
<h1>The h1 tags enclose a heading level 1</h1>
<p>The p tags is used to <em>markup</em> a <strong>paragraph</strong>.</p>
```

2. **Standalone Tag (aka Empty Tag, Bodyless Tag):** A standalone tag does not enclose content but is used to markup a certain effect, e.g., `<hr>` is used to draw a horizontal rule; `<br>` to introduce a manual line-break; and `<img>` for embedding an external image. Standalone tag should be closed with a trailing `'/'` in the opening tag (for XHTML/XML compliance). A space may be needed before the `'/'`. For examples:

```
<br />
<hr />
<img src="logo.gif" />
```

   Alternatively, you can also close a standalone tag with a matching closing tag (which is cumbersome but consistent in syntax to the container tag), e.g.,

```
<br></br>
<hr></hr>
<img src="logo.gif"></img>
```

Tags <html>, <head> and <body>

The <html>...</html> container tag defines the extent of an HTML document.

An HTML document has two sections: HEAD and BODY:

1. <head>...</head> container tag defines the HEAD section, which contains *descriptions* and *meta-information* of the HTML document. Browsers interpret these descriptions to properly display the body content.
2. <body>...</body> container tag defines the BODY section, which encloses the *content* that users actually see in the browser's window.

The HEAD Section and the <title> Tag

Inside the HEAD section, the <title>...</title> tag encloses the *title* for the page. You should use a meaningful title because:

- The title shows up at the title-bar of the browser window.
- The title shows up in bookmarks and history lists (the URL is stored if there is no title).
- Titles are used by search engines' to index your page.
- W3C Recommends that "at a minimum, every HTML document must at least include the descriptive title element."

Other tags in the HEAD section include: <base>, <link>, <meta>, <script>, and <style>, which shall be discussed later.

Tag's Attributes in *name*="*value*" pairs

Attributes can be included in the *opening tag* to *provide additional information* about the element. For example, an <img> (image) tag has the following attributes:

```
<img src="logo.gif" width="50" height="30" />
```

Attributes "src", "width" and "height" are used in the <img> tag to specify the source-url, and width/height of the image displayed area. Some of the attributes are *mandatory* (e.g., the "src" attribute of the <img> tag); while some are *optional* (e.g., the "width" and "height" attributes of the <img> tag, which are used by browser to reserve space for the image; but browser can figure out the width and height after the image is loaded).

Attributes are written in the form of *name*="*value*" pairs. The *value* shall be enclosed in single or double quotes, for XHTML/XML compliance. The *name*="*value*" pairs are separated by space.

```
<tag-name attName1="attValue1" attName2="attValue2" ...> ... </tag-name>
```

The <body> Tag

The <body>...</body> tag defines the BODY section of an HTML document, which encloses the content to be displayed on the browser's window.

The <body> tag has the following optional presentation attributes. All of these presentation attributes are concerned about the appearance instead of the content, and have been deprecated in HTML 4 in favor of style sheet. However, many older textbooks present them in Chapter 1. Hence, I shall list them here for completeness. BUT do not use these attributes. I shall describe how to control the appearance of <body> using CSS very soon.

- text="*color*": color of body text.
- bgcolor="*color*": background color.
- background="*url*": URL of an image to be used as the background.
- link="*color*": color of un-visited links.
- vlink="*color*": color of visited links.
- alink="*color*": color of active (clicked) links.

For example:

```
<html>
<body text="blue" bgcolor="lightblue" link="green" vlink="red" alink="yellow">
  <p>Hello</p>
  <a href="http://www.google.com">Google</a>
</body>
</html>
```

The foreground color (of the texts) is "blue", on background color of "lightblue". You can set different colors for the three types of links via attributes "link" (for un-visited links), "vlink" (for visited links), and "alink" (for active link - the "alink" color shows up when you click on the link).

3.3 Basic HTML Tags

There are two types of tags (aka elements in XHTML/XML terminology):

1. **Block Elements (Block-Level Tags):** A block element (such as , <h1> to <h6> and) starts on a new line, takes the full width, and ends with a new line. It is *rectangular* in shape with a *line-break* before and after the element.
2. **Inline Elements (or Character-Level Tags or Text-Level Tags):** An inline element (such as , , <code> and) takes up as much space as it needs. It does not force a line-break before and after the element, although it can span a few lines.

In brief, a block element is always *rectangular* in shape, while a inline element spans *a continuous run of characters*.

3.4 Block-Level Tags (aka Block Elements)

You should have an online HTML reference such as W3School's HTML/XHTML Reference @ http://www.w3schools.com/tags/default.asp; or the "HTML 4.01 Specification (1999)" (@ http://www.w3.org/TR/html401/) and "XHTML 1.0 Specification (2002)" (@ http://www.w3.org/TR/html/) to read the following sections, as I do not intend to list out every details.

Paragraph ...

Function: When the browser reads a tag, it breaks to a new line, and skips some white spaces. For example,

```
<p>This is a paragraph of texts.</p>
```

Older HTML documents often omit the closing , which is a bad practice, not recommended, and disallowed in XML/XHTML.

Line-Break

Function: Instruct the browser to break to a new line, without skipping white spaces as in . Note that the line breaks in the HTML codes are treated as white spaces and do not translate to new lines in the display. Hence, you have to insert the <br /> manually. For example,

```
<p>This
paragraph<br />with a
line-break
in between.</p>
```

This paragraph
with a line-break in between.

Heading Level 1 to 6 <h1>...</h1> to <h6>...</h6>

Function: Establish six levels of document headings. <h1> is the highest (with the largest font size) and <h6> is the lower. Headings are usually displayed in bold, have extra white spaces above and below. For example,

```
<h1>This is Heading Level 1</h1>
<h2>This is Heading Level 2</h2>
<h3>This is Heading Level 3</h3>
<h4>This is Heading Level 4</h4>
<h5>This is Heading Level 5</h5>
<h6>This is Heading Level 6</h6>
<p>This is a paragraph</p>
```

Horizontal Rule <hr />

Function: Draw a horizontal line (or rule). By default, the rule is full width (100%) across the screen, 1 point in size, and has a shading effect for a 3D appearance. For example,

```
<h1>Heading</h1>
<hr />
<p>Discussion begins here...</p>
```

Pre-formatted Text <pre>...</pre>

Function: Texts enclosed between <pre>...</pre> container tag are treated as *pre-formatted*, i.e., white space, tabs, new-line will be preserved and not ignored. The text is usually displayed in a fixed-width (or monospace) font. <pre>...</pre> is mainly used to display program codes. For example, my favorite Java's "Hello-world":

```
<pre>public class Hello {
   public static void main(String[] args) {
      System.out.println("Hello");
   }
}</pre>
```

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

Without the <pre> tag, the entire program will be shown in one single line.

3.7 Entity References for Reserved & Special Characters

HTML uses characters such as <, >, ", & as markup tags' delimiters. Hence, these characters are reversed and cannot be used in the text directly. An escape sequence (called *entity reference*), in the form of &*xxx*; (begin with & and end with ;) is used for these reserved characters and other special characters. The commonly used entity references are as follows (there are many many more, refer to the HTML reference - I like the arrows, greek symbols, and the mathematical notations). Entity reference is case sensitive.

Character	Entity Reference
"	&quot;
<	&lt;
>	&gt;
&	&amp;
non-breaking space	
→ ⇒ ↔ ⇔	→ ⇒ ↔ ⇔
° (degree)	&deg;
© ® € ¢ ¥	© ® € ¢ ¥
~	˜
× ± ∞	× ± ∞
π ∏ σ Σ ω Ω	π Π σ Σ ω Ω
≥ ≤ ≡ ≈	≥ ≤ ≡ ≈
⊂ ⊃ ⊆ ⊇ ∈	⊂ ⊃ ⊆ ⊇ ∈

You need to memorize the first five: " ("), < (<), > (>), & (&) and ( ).

Non-breaking Space ( )

As mentioned earlier, browsers ignore extra whitespaces (blanks, tabs, newlines). That is, multiple whitespaces are treated as one single whitespace. You need to use the non-breaking spaces ( ) to insert multiple whitespaces.

Example

```
<p>This paragraph contains special character &quot; &lt;, &gt; and &amp;
and those    words    have   
more    spaces in between.</p>
```

This paragraph contains special character " <, > and & and those words have more spaces in between.

3.8 Lists - Ordered List, Unordered List and Definition List

List-related tags are meant for marking up *a list of items*. HTML supports three types of lists: *ordered list*, *unordered list* and *definition list*.

Unordered List `<ul>...</ul>` and List Item `<li>...</li>`

Function: An unordered list is shown with a *bullet* in front of each item. The `<ul>...</ul>` contains an unordered list. Each of items in the list is enclosed in `<li>...</li>`, as follow:

```
<ul>
  <li>list-item-1</li>
  <li>list-item-2</li>
  ......
</ul>
```

Example:

```
<p>An OMO web designer must master:</p>
<ul>
   <li>Hypertext Markup Language (HTML)</li>
   <li>Cascading Style Sheet (CSS)</li>
   <li>HyperText Transfer Protocol (HTTP)</li>
   <li>Apache HTTP Server</li>
</ul>
```

Output of the example:

An OMO web designer must master:

- Hypertext Markup Language (HTML)
- Cascading Style Sheet (CSS)
- HyperText Transfer Protocol (HTTP)
- Apache HTTP Server

3.9 Tables

Table-related tags are meant for tabulating data. (Older HTML documents tend to use `<table>` for formatting the document to divide the document into columns/sections, which should be avoid. Use style sheet for formatting instead.)

The basic unit of a table is a *cell*. Cells are grouped into *row*. Rows are grouped to form the *table*. This corresponds well to the "row-centric" approach in the display.

The tags used by table are:

- `<table>...</table>`: contains the entire table.
- `<tr>...</tr>`: contains a row.
- `<th>...</th>` and `<td>...</td>`: contain a *header* cell and a *data* (*detail*) cell respectively.
- `<caption>...</caption>`: specifies a caption.
- `<thead>...</thead>`, `<tbody>...</tbody>`, and `<tfoot>...</tfoot>`: for table header, body, and footer.
- `<colgroup>...</colgroup>` and `<col>...</col>`: for applying styles to column group and column respectively.

For Example:

```
<table>
  <caption>Price List</caption>
  <tr>
    <th>Fruit</th>
    <th>Price</th>
  </tr>
  <tr>
    <td>Apple</td>
    <td>$0.50</td>
  </tr>
  <tr>
    <td>Orange</td>
    <td>$0.65</td>
  </tr>
</table>
```

Price List

Fruit	Price
Apple	$0.50
Orange	$0.65

Table Row `<tr>...</tr>`

Function: Set up a row inside a table, consisting of cells.

Table Header Cell `<th>...</th>`, Table Data Cell `<td>...</td>`

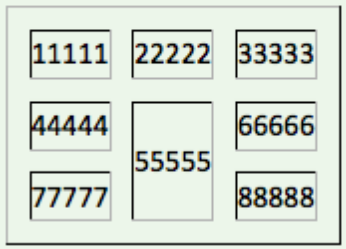
Function: Set up each individual cell of a row (of a table). `<th>...</th>` defines a header cell (usually displayed in bold with center alignment) and `<td>...</td>` defines a body cell.

Attributes "rowspan" and "colspan"

On a `<td>` cell, we can use the attribute `rowspan="numOfRows"` or `colspan="numOfColumns"` to span the cell to occupy multiple rows or columns. The subsequent `<td>` cells will adjust their positions accordingly.

Example 1:

```
<table>
  <tr>
    <td>11111</td>
    <td>22222</td>
    <td>33333</td>
  </tr>
  <tr>
    <td>44444</td>
    <td rowspan="2">55555</td>
    <td>66666</td>
  </tr>
  <tr>
    <td>77777</td>
    <td>88888</td>
  </tr>
</table>
```



3.10 Anchor and Link <a>...

A hypertext-link (or hyperlink, or link) allows users to:

1. navigate to a different document.
2. navigate to an "Anchor Point" (bookmark) in the current document or another document, or
3. request other Internet resources (such as e-mail).

The anchor tag <a>... can perform one of these two functions:

1. It can be used to set up a *hyperlink*, where user can navigate to the target document by clicking the link.
2. It can also be used to set up a "*named anchor point*" (bookmark) within a document, to be targeted by other hyperlinks. Named anchors are useful for long documents.

Hyperlink (Hypertext Reference) ...

Function: to set up a hyperlink pointing to *url* in "href" (*hypertext reference*).

Examples:

```
<a href="http://www.w3c.org">W3C Home Page</a>
<a href="ftp://ftp.faqs.org">FTP to FAQS.ORG</a>
Show <a href="../images/logo.gif">LOGO</a>Image
<a href="mailto:help@zzz.com">Email Help</a>
<a href="news:soc.culture.singapore">Singapore News</a>
```

Link's Target Window/Frame ...

Instead of displaying the targeted page pointed to by href in the *current* browser's window. You can use the attribute target="*targetName*" to display the new page in another window. For example, target="_blank" opens the new page in a new blank tab.

[Note: The target attribute has been deprecated in HTML 4.01 and XHTML 1.0. But it seems to be back in HTML 5. You could use it freely, as it is supported by most browsers.]

URLs (Uniform Resource Locators)

A URL uniquely identifies a piece of resource over the Internet. A URL is made up of 4 parts as follows:

```
Protocol://hostname:port/path_and_filename
```

1. Protocol: e.g., `http`, `ftp`, `mailto`, `file`, `telnet` and others.
2. Server hostname (e.g., www.w3c.org) or IP address (e.g., 127.0.0.1). The DNS (Domain Name Service) translates a domain name to an IP address.
3. Port number (optional): the TCP port number on which the server application is running. The default TCP port number is used if port number is omitted from the URL. For example, default TCP port number 80 will be used for HTTP, 21 for FTP.
4. Directory path and file name: Unix-style forward-slash `'/'` is used as the path separator (instead of Windows-style back-slash `'\'`). Directory path and filename of the URL are case sensitive.

Examples of URLs are:

```
http://www.w3c.org/css/index.html
http://www.mytest.com:8080/default.html
ftp://ftp.faqs.org
news:soc.culture.singapore
mailto:help@zzz.com
```

Absolute vs. Relative URLs

An absolute URL is *fully qualified*, e.g., `http://www.mytest.com/abc/index.html`. A relative URL is *relative* to a *base URL*.

For example, suppose that the base URL is `http://www.mytest.com/abc/index.html`, the *base path* (excluding the filename) is `http://www.mytest.com/abc/`.

- The relative URL "`test.html`" refers to `http://www.mytest.com/abc/test.html`.".
- The relative URL "`../home.html`" refers to `http://www.mytest.com/home.html`, where the double dot "`..`" denotes the parent directory.

Rule of Thumb: Always use relative URLs for referencing documents in the same server for portability (i.e., when you move from a test server to a production server). Use absolute URLs only for referencing resources from a different server.

3.11 Using Images

Image Tag <img>

Function: Embed an image inside an HTML document. The syntax is:

```
<img src="imageUrl"
  alt="alternate-text-if-image-cannot-be-shown"
  width="pixels|n%"
  height="pixels|n%"
  title="tooltip-text" />
```

Example:

```
<img src="logo.gif" width="200" height="100">
```

Attributes:

- src="*imageUrl*": mandatory, gives the URL of the image.
- alt="*text*": alternative text to be displayed if the image cannot be displayed.
- width="*n*|*n*%", height="*n*|*n*%": specify the width and height of the image display area (in pixels or percentage). Browsers use these values to *reserve* space for the image (before the image is downloaded) and continue rendering the rest of the contents. You can also use the width and height to *scale* an image. This is not recommended because scaling-up results in a blur image and scaling-down is a waste of bandwidth.
  I recommend that you use the width and height tags for images, so that the browser can reserve spaces for the images. This is more efficient and could avoid a *jerky* display if you page contains many images. You can find out the width and height of an image easily by checking the "Properties" of the image.
- title="*tooltip-text*": the attribute title is applicable to most of the HTML tag for you to provide the tool tip text.

Image as Hyperlink

To use an image as a hyperlink, put the image tag <img> between and . For example:

```
<a href="http://abc.com/">
  <img src="logo.gif" alt="logo Image" width="30" height="20">
</a>
<p>click the above image to visit us</p>
```

Image used as hyperlink anchor automatically gets a border. The color of the border is given in the link (unvisited), vlink (visited), alink (active) attributes of the <body> tag (or the a:link, a:visited, a:hover, and a:active CSS properties - to be discussed later).

CSS

https://www3.ntu.edu.sg/home/ehchua/programming/index.html

4. Introduction to CSS

4.1 Why Style Sheet?

The *original* aim of HTML is to let the content providers concentrate on the content of the document and leave the appearance to be handled by the browsers. Authors markup the document content using markup tags (such as `<p>`, `<h1>`, `<ul>`, `<table>`, `<img>`) to indicate its *semantic meaning* ("This is a paragraph", "This is heading Level 1", "This is an unordered list", "This is a table", "This is an image"). The browser then decides on how to *display* or *present* the content on the browser's window.

However, HTML has gone out of control in the early years. Tons of markup tags and attributes were created for marking the appearance and the display styles (e.g., `<font>`, `<center>`, `align`, `color`, `bgcolor`, `link`, `alink`, `vlink` are concerned about the appearance in font, color and alignment) rather than the meaning of the content. It is important to separate the content and presentation of a document. Document contents are provided by the document authors or content providers, whereas presentation is usually done by a graphic designers.

The W3C (World-Wide Web Consortium @ www.w3c.org) responded to the need of separating document's content and presentation by introducing a style sheet language called CSS (Cascading Style Sheet). CSS can be viewed as an *extension* (or *companion*) of HTML, which allows web graphic designer to spice up the web pages, so that the content providers can focus on the document contents.

There are two CSS levels:

1. CSS Level 1 (1996): CSS1 laid the ground work, introduces the selectors and most of the properties.
2. CSS Level 2 (1998) / 2.1 (2006): CSS2 added new features such as targeting devices and printers, and absolute positioning. CSS2.1 (@ http://www.w3.org/TR/CSS2/) lightly touch-up CSS2.
3. CSS Level 3: Still in progress.

CSS is humongous!!! Most of the browsers today have yet to fully support CSS 2, which was published in 1998 and lightly updated to Level 2.1 in 2006?!

4.2 What is a Style Sheet?

A *Style Sheet* is *a collection of style rules* that can be applied to *a selected set of HTML elements*. A style rule is used to control the appearance of HTML elements such as its font properties (e.g., type face, size, weight), color properties (e.g., background and foreground colors), alignment, margin, border, padding, and positioning. This is the same as the style in any publishing software like WinWord or LaTex.

The word "cascading" means that multiple style rules can be applied to the same HTML element simultaneously. The browser follows a certain "cascading order" in finalizing a style to format an HTML element in a predictable fashion.

4.3 A Sample CSS Style Sheet

```
/* A CSS Style Sheet - This is a comment */
body {
  font-family: "Segoe UI", Tahoma, Helvetica, Arial, Verdana, sans-serif;;
  font-size: 14px;  /* in pixels (px) */
  margin: 10px, 0, 10px, 0;  /* top right bottom left */
  padding: 0
}
h1, h2, h3, h4, h5, h6 {
  font-family: "Trebuchet MS", "Segoe UI", Helvetica, Tahoma, Arial, Verdana, sans-serif;
  color: red;
  background: black;
  font-style: italic;
  font-weight: bold;
  text-align: center;
}
p {
  text-align: justify;
  font-size: 14px;
  color: #000000;  /* black */
}
```

4.4 CSS Syntax

CSS is a language by itself. It has its own syntax, which is totally different from HTML! (How many syntaxes you have to know to program the web?!).

A CSS style sheet is a *collection of style rules*:

1. A *style rule* consists of a *selector* which selects the HTML elements it operates upon (e.g., <body>, <p>, <ul>), and a list of style property names and values enclosed in braces { }, as follows:

   ```
   selector {
     property-name-1: property-value-1-1, property-value-1-2, ... ;
     property-name-2: property-value-2-1, property-value-2-2, ... ;
     ......
   }
   ```

   For example,

   ```
   body {   /* Apply to <body> and possibly its descendants */
     font-family: "Segoe UI", Tahoma, Helvetica, Arial, Verdana, sans-serif;;
     font-size: 14px;
     margin: 10px, auto, 10px, auto;  /* top right bottom left */
     padding: 0;
   }
   ```

   The *selector* selects the <body> tag. Hence, the defined style is applied to the <body>...</body> element. Many (but not all) of the CSS properties (such as color, font) are *inherited* by the descendents, unless it is overridden by another style definition.

2. The property *name* and *value* are separated by a colon ':' in the form of *name*:*value*. Multiple values are separated by commas ',' or whitespace. Values containing space must be quoted, e.g., "Times New Roman" or 'Times New Roman'.

3. The property *name*:*value* pairs are separated by semicolon ';'. You can omit the last semi-colon ';' before the closing brace '}'. But I recommend that you keep it, so that it is easier to include new entries without a missing ';'.

4. Extra whitespaces (blank, tab and newline) are ignored.

5. If the same styles are applicable to more than one tags, the selectors can be grouped together in one single rule, separated by commas ','. For example, the following rule apply to HTML tags <h1> to <h6>:

   ```
   h1, h2, h3, h4, h5, h6 {
     text-align: center;
     font-family: "Trebuchet MS", "Segoe UI", Helvetica, Tahoma, Arial, Verdana, sans-serif;
   }
   ```

6. Comments can be inserted inside the sheet sheet enclosed between /* and */. (The end-of-line comment // does not work in CSS?!)

Take note that CSS and HTML have different syntaxes. In HTML, tags' attributes uses '=' to separate the *name* and *value* pair, in the form of *name*="*value*"; attributes are separated by spaces. For example,

```
<img src="logo.gif" height="10" width="20">
```

4.5 Types of Styles

There are three places where you can define style rules:

1. **Inline Style**: included inside a particular HTML tag's attribute `style="style-rules"`, and is applicable to that particular HTML element only.
2. **Embedded Style Sheet**: embedded inside the `<style>...</style>` tag in the HEAD section of the HTML document. The styles are applicable to that document only.
3. **External Style Sheet (Recommended)**: stored in an external file, which is then linked to HTML documents, via a `<link>` tag in the HEAD section. An external style sheet can be applied to many HTML documents to ensure uniformity (e.g., all pages in your website).

Inline Styles

To apply inline style to an HTML element, include the list of style properties in the `style` attribute of the tag. For example,

```
<!DOCTYPE html>
<html>
<body>
  <p style="font-size:18px; font-family:cursive">This paragraph uses 18px cursive font.</p>
  <p>This paragraph uses default font.</p>
  <p>This paragraph uses <span style="font-size:20px">20px inside this span</span>
  but default font size here.</p>
</body>
</html>
```

This paragraph uses 18px cursive font.

This paragraph uses default font.

This paragraph uses 20px inside this span but default font size here.

The scope of an inline style is limited to that particular tag. Inline style defeats the stated goal of style sheets, which is to separate the document's content and presentation. Hence, inline style should be avoided and only be used sparsely for *touching up a document*, e.g., setting the column width of a particular table.

Embedded Styles

Embedded styles are defined within the `<style>...</style>` tag in the HEAD section. For example,

```
<!DOCTYPE html>
<html>
<head>
  <style type="text/css">
     body      { background-color:cyan }
     h2        { color:white; background-color:black }
     p.cursive { font-size:18px; font-family:cursive }
     p.f20px   { font-size:20px }
  </style>
</head>
<body>
  <h2>H2 is white on black</h2>
  <p>This paragraph is normal.</p>
  <p class="cursive">This paragraph uses 18-px cursive font.</p>
  <p class="f20px">This paragraph uses 20-px font.</p>
</body>
</html>
```

- The scope of the embedded styles is the current HTML document.
- Embedded styles separate the presentation and content (in the HEAD and BODY sections) and can be used if page-to-page uniformity is not required. That is, this set of styles is used for only one single page!? (I use embedded style in this article for illustration, but you should use external style in production.)
- The attribute `type="text/css"` in the `<style>` opening tag is not needed in HTML 5.

External Style Sheets

External styles are defined in a external file, and referenced in an HTML document via the `<link>` tag in the HEAD section.

For example, we define these style rules in a file called "`testExternal.css`":

```
/* testExternal.css */
body    { background-color:cyan; color:red; }
h2      { background-color:black; color:white; text-align:center; }
p       { font-size:12pt; font-variant:small-caps; }
p.f24pt { font-style:italic; font-size:24pt; text-indent:1cm; }
#green  { color:green; }
```

This HTML document references the external style sheet via the `<link>` tag in the HEAD section:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" type="text/css" href="testExternal.css">
</head>
<body>
  <h2>H2 is white on black</h2>
  <h2 id="green">This H2 is green on black</h2>
  <p>The default paragraph uses 12-pt small-cap font.</p>
  <p class="f24pt">This paragraph uses 24-pt, italics font with text-indent of 1cm.
  It inherits the small-cap property from the default paragraph selector.</p>
</body>
</html>
```

The main advantage of external style sheet is that the same styles can be applied to many HTML pages to ensure *uniformity* in presentation for your entire website. External style sheet is the now-preferred approach.

Note: HTML 5 does not requires the `type="text/css"` attribute in `<link>` tag.

4.6 Using CSS for Styling

To use CSS to style your website and to get the full benefit of CSS, you need to properly structure your web pages.

HTML Division `<div>` and Span `<span>` Tags

Two HTML tags, division `<div>...</div>` and span `<span>...</span>` are primarily designed for applying CSS styles. They can be used to create *partitions* in an HTML document to represent logical sections (such as header, content, footer). `<div>` is a *block element* which is rectangular in shape; while `<span>` is an *inline element* which spans a sequence of characters.

Modern-day HTML pages use `<div>` and `<span>` extensively to layout the document via CSS. (Older HTML pages uses tables and frames, which should be avoided. HTML 5 introduces new tags such as `<header>`, `<footer>`, `<section>`, `<nav>`, `<article>` to help you better organize your page.)

You can treat `<div>` and `<span>` as *generic* tags for identifying contents, which shall be further qualified via the `id` or `class` attribute. Similarly, `<div>` and `<span>` does not have any inherit visual properties (unlike `<h1>`), you need to attach presentation properties via CSS.

First of all, partition your web page in *logical sections* via `<div>...</div>`, such as header, content, navigation menu, footer, etc.

CSS Selector

As mentioned earlier, a CSS *selector* is used to select a group of HTML elements to apply the defined styles. The selection can be carried out via:

1. HTML tag name, e.g., body, p, `ol`, `table`, and `div`.
2. HTML tag's `id` attribute, e.g., `<div id="header">`.
3. HTML tag's `class` attribute, e.g., `<span class="highlight">`.
4. others.

HTML Tag Attributes `id="idValue"` and `class="classname"`

All the HTML tags supports two optional attributes: `id="idValue"` and `class="className"`.

1. You can assign an `id="idValue"` to an HTML element to uniquely identify that element. The `id` value must be unique within the HTML document. In other words, no two elements can have the same `id` value. The `id` attribute is used by CSS as well as JavaScript to select the element. For example,

```
<div id="header"><h1>Header Section</h1></div>
<div id="content"><h1>Content Section</h1></div>
<div id="footer"><h1>Footer Section</h1></div>
```

2. The `class` value needs not be unique. That is, the same `class` value can be assigned to many HTML elements. In other words, these HTML elements form a *sub-class*. The `class` attribute is primarily used by CSS to apply a common set of styles to all the elements of the same `class`. (HTML 5 can also selects elements based on `class` name). For example,

```
<p class="highlight">A highlighted paragraph</p>
<p class="highlight">Another highlighted paragraph</p>
```

3. A `class` may contain multiple values, separated by space, e.g.,

```
<p class="highlight underline">This paragraph element has two class values</p>
```

Partition Your Web Page into Logical Sections

The `id` and `class` attributes are important for to identify partitions created via `<div>` and `<span>`, for applying styles.

The first step in web page design is to partition your web page in logical sections using the `<div>...</div>` tag, and assign `id` or `class` to each of the divisions. Use `id` if the division is unique (in formatting); otherwise, use `class` (more than one divisions have the same formatting).

Example

The following HTML page is divided into three divisions, with unique `id`'s of "`header`", "`content`" and "`footer`" respectively. Selected texts are marked with the `<span>` tags, with `class` of "`green`" and "`blue`".

```
<html>
<head>
<style>
  /* style-definitions - see below */
</style>
</head>

<body>
<div id="header">
  <h1>Heading</h1>
</div>

<div id="content">
<h2>Hello</h2>
<p>Lorem ipsum dolor sit amet, <span class="green">consectetur adipisicing</span> elit, sed do
eiusmod <span class="green">tempor incididunt ut labore et dolore magna aliqua. Ut enim ad
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in</span> reprehenderit in voluptate <span class="blue">velit esse</span>
cillum dolore eu fugiat nulla pariatur.</p>
</div>

<div id="footer">
  <p>Footer</p>
</div>
</body>
</html>
```

CSS Tag-name, Id and Class Selectors

Tag-name Selector (or Tag Selector or Element Selector): A CSS *tag-name selector* selects HTML elements based on the *tag-name*, e.g.,

```
/* tag-name selector */
h2 { background-color:black; color:white; text-align:center; }
```

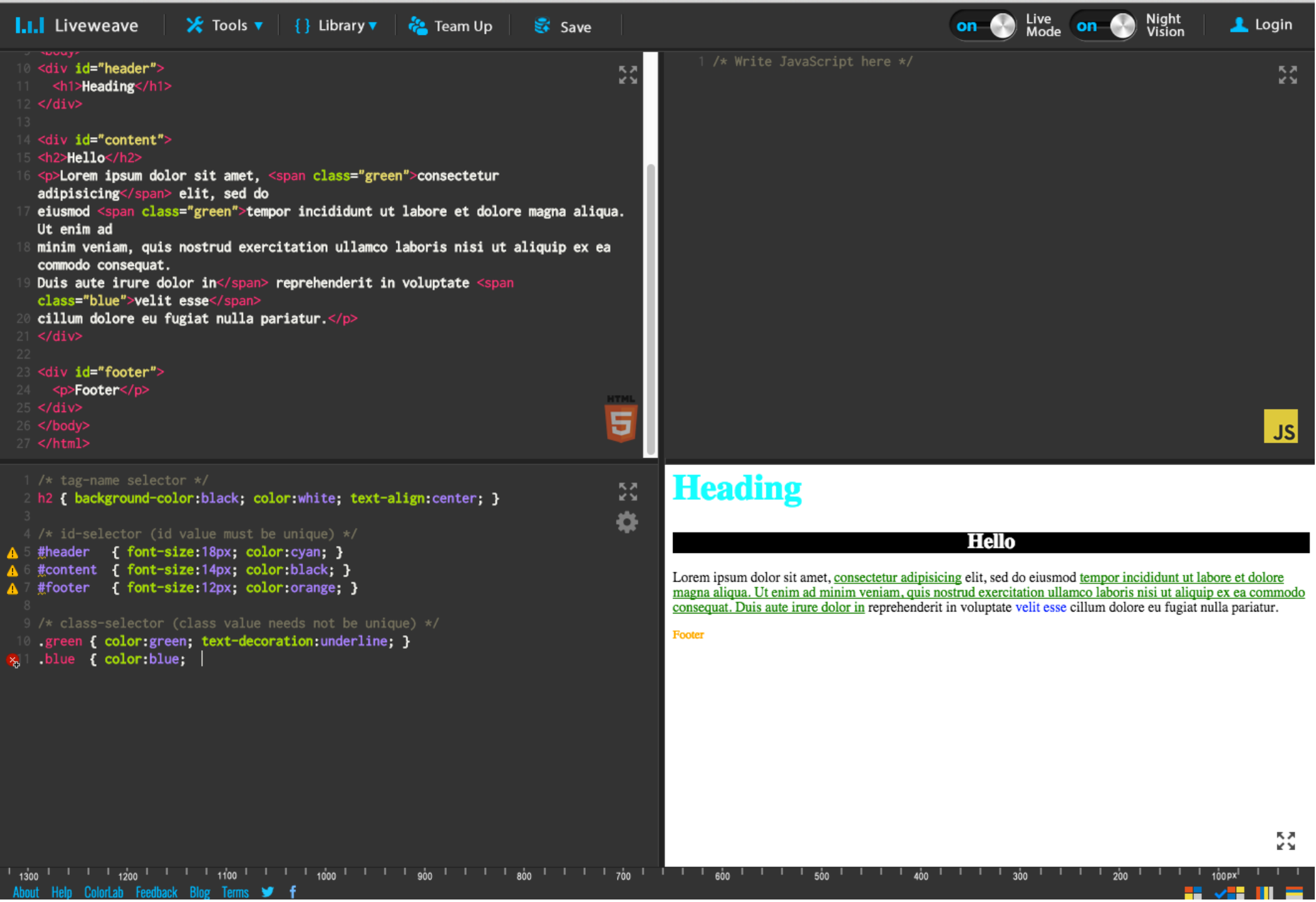
ID-Selector: You can use CSS to set the display style for each of the three divisions, via the so-called *id-selector*. A CSS *id-selector* begins with a '#' followed by the id's value. It selects a specific element. For example,

```
/* id-selector (id value must be unique) */
#header   { font-size:18px; color:cyan; }
#content  { font-size:14px; color:black; }
#footer   { font-size:12px; color:orange; }
```

Class Selector: CSS also provides the so-called *class-selector* for selecting elements of the same class value. A CSS *class-selector* begins with a '.' followed by the classname. For example,

```
/* class-selector (class value needs not be unique) */
.green { color:green; text-decoration:underline; }
.blue  { color:blue;  }
```

Note: Because of the wide popularity of using 's with id="header", id="footer" to organize an HTML page, HTML 5 defines new tags such as <header>, <footer>, <section>, <article>, <nav> to help you to better organize the HTML page.


Liveweave
Tools
Library
Team Up
Save
on
Live Mode
on
Night Vision
Login

<h1>Heading</h1>

<h2>Hello</h2>
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.

Footer

</body>
</html>
/* Write JavaScript here */
/* tag-name selector */
h2 { background-color:black; color:white; text-align:center; }
/* id-selector (id value must be unique) */
#header { font-size:18px; color:cyan; }
#content { font-size:14px; color:black; }
#footer { font-size:12px; color:orange; }
/* class-selector (class value needs not be unique) */
.green { color:green; text-decoration:underline; }
.blue { color:blue;
Heading
Hello
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur.
Footer
1300 1200 1100 1000 900 800 700 600 500 400 300 200 100px
About Help ColorLab Feedback Blog Terms

4.8 Types of CSS Selectors

As illustrated in the previous example, a CSS selector can select a set of HTML elements based on (a) tag name, (b) tag's `id` attribute, (c) tag's `class` attribute, and (d) their *many many* combinations. (Read "The 30 CSS Selectors you Must Memorize". CSS is really humongous!?)

I shall list the frequently-used types of selectors here.

`T` - Tag-Name Selector (aka Tag Selector, or Type Selector, or Element Selector)

A *tag-name selector* selects all elements of the given tag-name. The syntax is:

```
tag-name { style-definitions }
```

Example:

```
h2 { background-color:black; color:white; }
```

`S1, S2` - Group Selector

You can apply the same style definitions to multiple selectors, by separating the selectors with a commas ', '. The syntax is,

```
selector-1, selector-2, ... { style-definitions }
```

Example:

```
h1, h2, h3 { background-color:black; color:white; }
```

T1 T2 - Descendant Selector

You can define a style rule that takes effect only when a tag occurs within a certain *contextual* structure, e.g., descendent, child, first-child, sibling, etc.

To create a *descendant selector*, list the tags in their hierarchical order, with no commas separating them (commas are meant for grouping selectors). For example,

```
ul li { color:red; }
ul ul li { color:blue; }
ul ul ul li { color:green; }
```

The first-level list items are in red; second-level in blue; and third-level in green.

Note: In T1 T2 { ... }, T2 is a descendant of T1 regardless of the generation.

T1 > T2 - Child Selector

A contextual selector where T2 is an immediate child of T1.

T1:T2 - First-Child Selector

A contextual selector where T2 is the *first* child of T1.

T1 + T2 - Adjacent Selector (aka Sibling Selector)

The style is applied to tag T2, only if it follows immediately after tag T1. T1 and T2 are siblings in the same hierarchical level.

.c - Generic Class Selector

The *generic class selector*, which begins with a dot '.' followed by the classname, selects all elements with the given classname, regardless of the tag name. For example,

```
.f14px_i { font-size:14px; font-style:italic; }
.f16px_b { font-size:16px; font-weight:bold; }
.red { color:red; }
.underline { text-decoration:underline; }
```

```
<p class="f14px_i">Text is 14px and italic.</p>
<p class="f16px_b">Text is 16px and bold.</p>
<p class="red">Text is in red.</p>
<h5 class="red">Text is in red.</p>
```

Take note that the `class` attribute may contain multiple values. This means that you can apply multiple class style rules to an HTML tag. For example,

```
<p class="f14px_i underline">Text is 14px and italic, and underlined.</p>
<p class="f16px_b red underline">Text is 16px and bold, in red and underlined.</p>
```

T.C - Class Selector

The selector T.C selects all tag-name T with classname of C. This is a restricted form of the generic class selector, which applies to the specific tag-name only.

An HTML tag (such as <p>) can be sub-divided into different *style sub-classes* via the `class` attribute. This subclass mechanism allows us to apply different styles to different subclass of a particular tag.

For example,

```
p          { color:black; }   /* default style for all <p> tags */
p.red      { color:red;   }   /* applicable to <p class="red"> tags (override default) */
p.blue     { color:blue;  }   /* applicable to <p class="blue"> tags (override default) */
h1, h2, h3 { color:green; }   /* default style for <h1>, <h2> and <h3> tags */
h3.white   { color:white; }   /* applicable to <h3 class="white"> tags (override default) */
h3.upper   { text-transform:uppercase; }
```

```
<p>This paragraph is in black (default style)</p>
<p class="red">This paragraph, of class="red", is in red.</p>
<p class="blue">This paragraph, of class="blue", is in blue.</p>
<h2>H2 in green (default style)</h2>
<h3>H3 in green (default style)</h3>
<h3 class="white upper">This H3, of class="white", is in white</h3>
```

Note: Do NOT start a class-name with a number! (This is the same restriction for identifiers in most of the programming languages.)

* - selector (Universal Selector)

The * universal selector selects ALL the tags in the document. For example,

```
* { margin:0; padding:0; }  /* all tags have margin and padding of 0 */
```

#D - ID Selector

The id-selector, begins with a '#' followed by the id's value, select a specific element with the given unique id value (or none). Recall that the id value must be unique in an HTML document.

You can use 's with unique id to divide the document into parts of different styles. For example,

```
/* ID selector for the 3 major division of the document */
#header     { font-size:16px; align:center; font-style:bold; }
#header h1  { text-transform:uppercase; }                 /* contextual selector */
#content    { font-size:14px; align:justify; }
#content h3 { color:#FF0000; text-decoration:underline; } /* red, underline */
#footer     { font-size:12px; align:right; }
#footer p   { color:#00FF00; text-decoration:none; }      /* green, not underline */
```

```
<body>
<div id="header">
  <h1>H1 in the "header" division</h1>
  <h3>H3 in the "header" division</h3>
  <p>Paragraph in "header" division</p>
</div>
<div id="content">
  <h1>H1 in the "content" division</h1>
  <h3>H3 in the "content" division</h3>
  <p>Paragraph in "content" division</p>
</div>
<div id="footer">
  <p>Paragraph in "footer" division</p>
</div>
</body>
```

T#D - Tag's ID Selector

Same as above but only if the id is defined under the tag. Since id value is supposed to be unique, this serves more as proper documentation.

JavaScript

https://www3.ntu.edu.sg/home/ehchua/programming/index.html

1. Introduction

JavaScript is the most widely used client-side programming language on the web. It is:

- a small, lightweight, object-oriented, cross-platform, special-purpose scripting language meant to be run under a host environment (typically a web browser).
- a *client-side scripting language* to enrich web user-interfaces and create dynamic web pages (e.g., form input validation, and immediate response to user's actions).
- the engine that supports AJAX (Asynchronous JavaScript and XML), which generate renew interest in JavaScript.

JavaScript, originally called LiveScript, was created by Brendan Eich at Netscape in 1995. Soon after, Microsoft launched its own version of JavaScript called JScript. Subsequently, Netscape submitted it to ECMA (formerly *"European Computer Manufacturers Association"*, now *"Ecma International - European association for standardizing information and communication systems"*) for standardization, together with Microsoft's JScript.

The ECMA Specification is called "ECMA-262 ECMAScript Language Specification" (also approved as "ISO/IEC 16262"):

- First Edition (June 1997)
- Second Edition (August 1998)
- Third Edition (December 1999)
- Forth Edition - abandon due to political differences
- Fifth Edition, 5.1 Edition (June 2011): not finalized, working draft only.

Meanwhile, the Mozilla Project (@ https://developer.mozilla.org/en/JavaScript) continues to upgrade the JavaScript with these major versions:

- 1.0 (1996)
- 1.3 (1998): ECMA-262 1st edition compliance
- 1.5 (1999): ECMA-262 3rd edition compliance
- 1.6, 1.7:
- 1.8 (2008), Latest 1.8.5: ECMA-262 5th edition compliance (Firefox 4)

1.1 JavaScript vs. Java

Java is a *full-fledged general-purpose* programming language created by James Gosling at Sun Microsystems (now part of Oracle), released in Aug 1995. JavaScript is created by Brendan Eich at Netscape in 1995. Originally called LiveScript, it is a small and lightweight *special-purpose* language for writing client-side program running inside the web browser to create active user-interface and generate dynamic web pages. Java also supports client-side programming via the so-called Java *applets*.

JavaScript is not a general-purpose nor a stand-alone programming language (it has to be run inside the browser). It was originally called LiveScript and was renamed to JavaScript in an ill-fated marketing decision to try to capitalize on the popularity of Java language, when Netscape released it Navigator 2 in 1996 (Navigator 2 also runs the Java applets). Java and JavaScript are totally different languages for different programming purposes. However, in the early days, some efforts were made to adopt Java syntaxes and conventions into JavaScript, such that JavaScript *seems* to be a subset of Java. In reality, they have very little in common. But, if you know Java, you should find JavaScript easier to learn because of these common syntaxes.

1.2 What JavaScript Cannot Do

Remember that JavaScript is a client-side program that you downloaded from a server, and run inside the browser of your (client) machine. What to stop someone from writing a JavaScript that wipes out your hard disk, or triggers a denial-of-service attack to another server? As a result, for security purpose,

1. It cannot read file from the client's machine.
2. It can only connect to the server that it come from. It can read file from the server that it come from. It cannot write file into the server machine.
3. It cannot connect to another server.
4. It cannot close a window that it does not open.

3.1 Example 1: Functions `alert()` and `document.write()`

Let us write our first JavaScript to print the message "Hello, world".

Start with a new file and enter the following codes. Do not enter the line numbers, which is used to aid in explanation. Take note that:

- JavaScript is *case sensitive*. A `rose` is NOT a *ROSE* and is NOT a *Rose*.
- "Extra" white spaces (blanks, tabs and newlines) are ignored. That is, multiple white spaces is treated as a single blank character. You could use them liberally to make your program easier to read.

Save the file as "`JSEx1.html`" (or any filename that you prefer, with file extension of "`.html`" or "`.htm`"). Run the script by loading the HTML file into a JavaScript-enabled browser.

"JSEx1.html"

```
1  <html>
2  <head>
3    <title>JavaScript Example 1: Functions alert() and document.write()</title>
4    <script type="text/javascript">
5      alert("Hello, world!");
6    </script>
7  </head>
8  <body>
9    <h1>My first JavaScript says:</h1>
10   <script type="text/javascript">
11     document.write("<h2><em>Hello world, again!</em></h2>");
12     document.write("<p>This document was last modified on "
13         + document.lastModified + ".</p>");
14   </script>
15 </body>
16 </html>
```

3.2 Example 2: Variable, if-else and Functions `prompt()`, `confirm()`

This script prompts the user for his/her name, confirms the name, and prints a greeting message.

There are three kinds of pop-up *dialog* boxes in JavaScript, to interact with the users:

1. The `alert(`*`string`*`)` function puts the *`string`* on a pop-up box with a OK button. User needs to click the OK button to continue.
2. The `prompt(`*`string`*`, `*`defaultString`*`)` function puts the *`string`* on a pop-up box with OK and Cancel buttons. It returns the input entered by the user as a string; or a special value called `null` if the user hits the Cancel button.
3. The `confirm(`*`string`*`)` function puts the *string* on a pop-up box with with OK and Cancel buttons. It returns `true` if user hits the OK button; or `false` otherwise.

"JSEx2.html"

```
<html>
<head>
  <title>JavaScript Example 2: Variables and function prompt()</title>
  <script type="text/javascript">
    var username = prompt("Enter your name: ", "");
    if (confirm("Your name is " + username)) {
       document.write("<p>Hello, " + username + "!</p>");
    } else {
       document.write("<p>Hello, world!</p>");
    }
  </script>
</head>
<body>
  <p>Welcome to JavaScript!</p>
</body>
</html>
```

3.3 Example 3: Date Object and Conditional Statement

The following script creates a Date object and prints the current time.

"JSEx3.html"

```
<html>
<head>
  <title>JavaScript Example 3: Date object and Conditional Statement</title>
  <script type="text/javascript">
    var now  = new Date();        // current date/time
    var hrs  = now.getHours();    // 0 to 23
    var mins = now.getMinutes();
    var secs = now.getSeconds();
    document.writeln("<p>It is " + now + "</p>");
    document.writeln("<p>Hour is " + hrs + "</p>");
    document.writeln("<p>Minute is " + mins + "</p>");
    document.writeln("<p>Second is " + secs + "</p>");
    if (hrs < 12) {
      document.writeln("<h2>Good Morning!</h2>");
    } else {
      document.writeln("<h2>Good Afternoon!</h2>");
    }
  </script>
</head>
<body></body>
</html>
```

3.4 Example 4: Loop

The following script prompts the user for a multiplier, and prints the multiples of 1 to 100 using a for-loop.

"JSEx4.html"

```
<html>
<head>
  <title>JavaScript Example 4: Loop</title>
</head>
<body>
  <h2>Testing Loop</h2>
  <script type="text/javascript">
    var multiplier = prompt("Enter a multiplier: ");
    for (var number = 1; number <= 100; number++) {
      document.writeln(number * multiplier);
    }
  </script>
</body>
</html>
```

3.5 Example 5: User-defined Function and onclick

Besides the JavaScript built-in functions such as `alert()`, `prompt()`, `write()`, and `writeln()`, you can define your own functions. A function has a name and a body consisting of a set of JavaScipt statements that collectively performs a certain task. It may take zero or more argument(s) from the caller and return zero or one value back to the caller.

"JSEx5.html"

```
<html>
<head>
  <title>JavaScript Example 5: User-defined function and Event onclick</title>
  <script type="text/javascript">
    function openNewWindow() {
      open("JSEx1.html");
    }
  </script>
</head>
<body>
  <h2>Example on event and user-defined function</h2>
  <input type="button" value="Click me to open a new window"
         onclick="openNewWindow()" />
</body>
</html>
```

3.6 Example 6: Event Handlers: onload, onunload, onmouseover, onmouseout

JavaScript can be used to handle many types of events, in response to a user's action or browser's action. For example,

- `onload`: fires *after* browser loaded the page.
- `onunload`: fires *before* the page is removed, e.g. another page is to be loaded, refreshing this page, or the window is closing.
- `onmouseover` and `onmouseout`: fires when the user points the mouse pointer at/away from the HTML element.

"JSEx6.html"

```
<html>
<head>
  <title>JavaScript Example 6: Events onload and onunload</title>
  <script type="text/javascript">
    var msgLoad = "Hello!";
    var msgUnload = "Bye!";
  </script>
</head>
<body onload="alert(msgLoad)" onunload="alert(msgUnload)">
  <p>"Hello" alert Box appears <em>after</em> the page is loaded.</p>
  <p onmouseover="this.style.color='red'"
     onmouseout="this.style.color=''">Point your mouse pointer here!!!</p>
  <p>Try closing the window or refreshing the page to activate the "Bye" alert box.</p>
</body>
</html>
```

3.7 Example 6a: Separating HTML, CSS and JavaScript

The previous example works fine. You will find many such example in textbooks, especially the older textbooks. However, it has a big problem. All the HTML contents, CSS presentation styles and JavaScript programming codes are placed in a single file. For a small toy program, the problem is not serious. But when your program grows and if the HTML, CSS and JavaScript are written by different people, you will have a real challenge in maintaining the program.

Let's rewrite the example to place the HTML, CSS and JavaScript in three different files.

"JSEx6a.html"

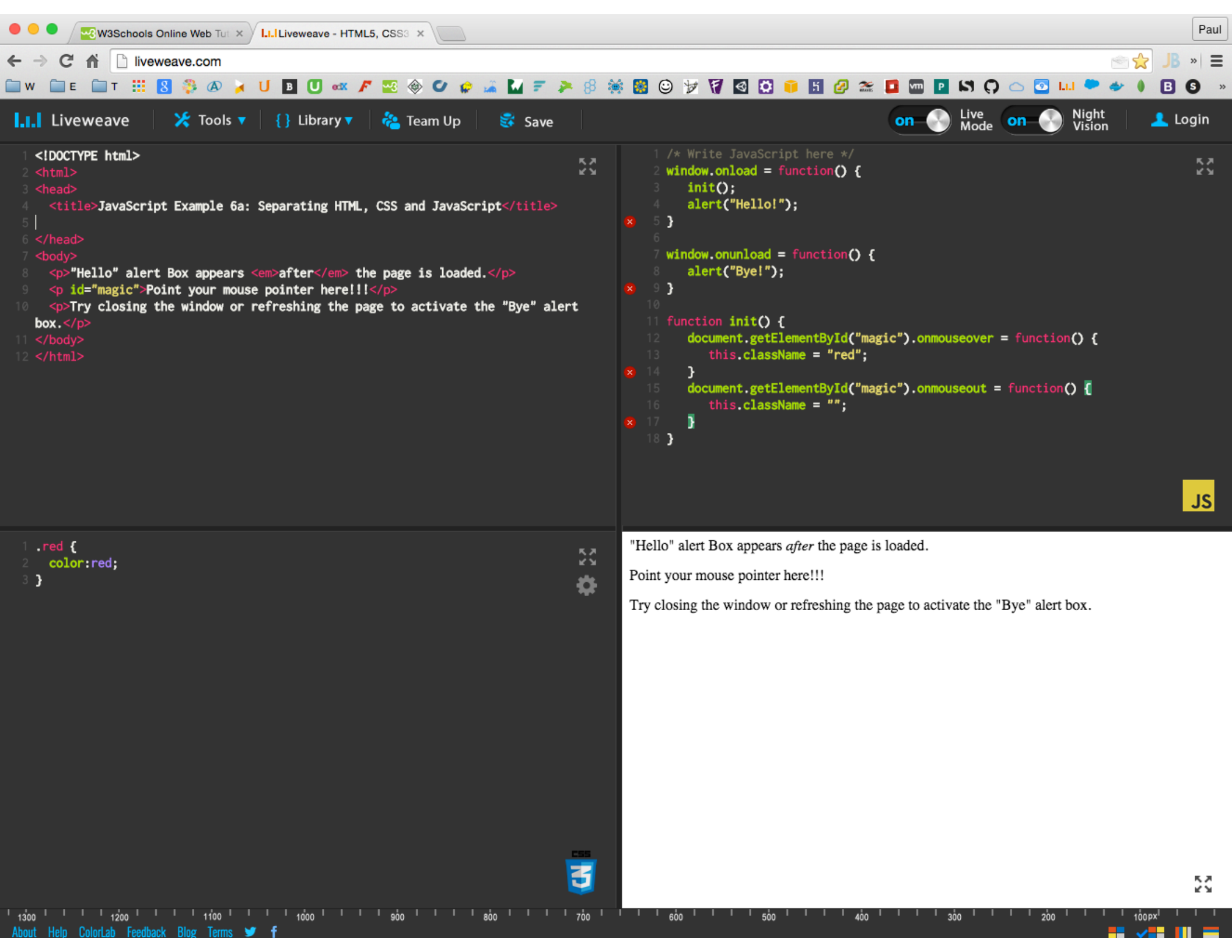
```
<html>
<head>
  <title>JavaScript Example 6a: Separating HTML, CSS and JavaScript</title>
  <link rel="stylesheet" href="JSEx6a.css">
  <script type="text/javascript" src="JSEx6a.js"></script>
</head>
<body>
  <p>"Hello" alert Box appears <em>after</em> the page is loaded.</p>
  <p id="magic">Point your mouse pointer here!!!</p>
  <p>Try closing the window or refreshing the page to activate the "Bye" alert box.</p>
</body>
</html>
```

"JSEx6a.css"

```
.red {
  color:red;
}
```

"JSEx6a.js"

```
window.onload = function() {
   init();
   alert("Hello!");
}

window.onunload = function() {
   alert("Bye!");
}

function init() {
   document.getElementById("magic").onmouseover = function() {
      this.className = "red";
   }
   document.getElementById("magic").onmouseout = function() {
      this.className = "";
   }
}
```


W3Schools Online Web Tu
Liveweave - HTML5, CSS3
Paul
liveweave.com
Liveweave
Tools
Library
Team Up
Save
on
Live Mode
on
Night Vision
Login
<!DOCTYPE html>
<html>
<head>
<title>JavaScript Example 6a: Separating HTML, CSS and JavaScript</title>
</head>
<body>
"Hello" alert Box appears after the page is loaded.
Point your mouse pointer here!!!
Try closing the window or refreshing the page to activate the "Bye" alert box.
</body>
</html>
/* Write JavaScript here */
window.onload = function() {
init();
alert("Hello!");
}
window.onunload = function() {
alert("Bye!");
}
function init() {
document.getElementById("magic").onmouseover = function() {
this.className = "red";
}
document.getElementById("magic").onmouseout = function() {
this.className = "";
}
}
JS
.red {
color:red;
}
CSS
"Hello" alert Box appears after the page is loaded.
Point your mouse pointer here!!!
Try closing the window or refreshing the page to activate the "Bye" alert box.
1300
1200
1100
1000
900
800
700
600
500
400
300
200
100px
About
Help
ColorLab
Feedback
Blog
Terms

3.8 Example 7: Modifying the Contents of HTML Elements and External Script

You can select HTML element(s) within the current page via these functions:

1. `document.getElementById(anId)`: returns the HTML element with `id="anId"`, or `null` if the `id` does not exist. The `id` attribute should be unique within an HTML document.
2. `document.getElementsByName(aName)`: returns an array of HTML elements with `name="aName"`. Take notes of the plural elements. More than one elements can have the same `name` attribute.
3. `document.getElementsByTagName(aTagName)`: returns an array of HTML elements with the given HTML tag name.

To modify the content of an HTML element, you can assign a new value to the `innerHTML` property of that element. (The property `innerHTML` is really useful and is supported in most of the browsers. It is, however, not included in the W3C DOM specification?!)

"JSEx7.html"

```
<html>
<head>
  <title>JavaScript Example 7: Modifying the Content of HTML Elements</title>
  <script type="text/javascript" src="JSEx7.js" ></script>
</head>
<body>
  <h1 id="heading1">Heading 1</h1>
  <h2>Heading 2</h2>
  <h2>Heading 2</h2>
  <p name="paragraph">Paragraph 1</p>
  <p name="paragraph">Paragraph 2</p>
  <p name="paragraph">Paragraph 3</p>
  <input type="button" id="btn1" value="Change Heading 1" />
  <input type="button" id="btn2" value="Change Heading 2" />
  <input type="button" id="btn3" value="Change Paragraph" />
</body>
</html>
```

"JSEx7.js"

```
1  window.onload = init;
2
3  function init() {
4    document.getElementById("btn1").onclick = changeHeading1;
5    document.getElementById("btn2").onclick = changeHeading2;
6    document.getElementById("btn3").onclick = changeParagraph;
7  }
8
9  function changeHeading1() {
10   document.getElementById("heading1").innerHTML = "Hello";
11 }
12
13 function changeHeading2() {
14   var elms = document.getElementsByTagName("h2");
15   for (var i = 0; i < elms.length; i++) {
16     elms[i].innerHTML = "Hello again!";
17   }
18 }
19
20 function changeParagraph() {
21   var elms = document.getElementsByName("paragraph");
22   for (var i = 0; i < elms.length; i++) {
23     elms[i].innerHTML = "Hello again and again!";
24   }
25 }
```

REFERENCES & RESOURCES

1. ECMAScript Specification: "Standard ECMA-262 ECMAScript Language Specification 5.1", (same as "ISO/IEC 16262" 3rd eds).
2. Mozilla's (MDN) JavaScript Project @ https://developer.mozilla.org/en/JavaScript. "Core JavaScript Guide" @ https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide, and "Core JavaScript Reference" @ https://developer.mozilla.org/en/JavaScript/Reference.
3. "Document Object Model (DOM)" Level 1, 2, 3 @ http://www.w3.org/standards/techs/dom.
4. "JavaScript" and "HTML DOM" Tutorials @ http://www.w3schools.com.

JDBC

https://www3.ntu.edu.sg/home/ehchua/programming/index.html

An Introduction to Java Database (JDBC) Programming by Examples

1. Relational Database and and Structure Query Language (SQL)

I presume that you have some knowledge on Relational Database and SQL. Otherwise, read "Introduction to Relational Database and SQL",

2.2 Install MySQL JDBC Driver

You need to install an appropriate JDBC (Java Database Connectivity) driver to run your Java database programs. The MySQL's JDBC driver is called "MySQL Connector/J" and is available at MySQL mother site. MySQL has a native JDBC driver, supporting many of the Java's JDBC features.

For Windows

1. Download the latest MySQL JDBC driver from http://dev.mysql.com/downloads ⇒ "MySQL Connectors" ⇒ "Connector/J" ⇒ Connector/J 5.1.{*xx*} ⇒ select "Platform Independent" ⇒ ZIP Archive (e.g., "`mysql-connector-java-5.1.{xx}.zip`", where {*xx*} is the latest release number).
2. UNZIP the download file into any temporary folder.
3. Copy the JAR file "`mysql-connector-java-5.1.{xx}-bin.jar`" to your JDK's Extension Directory at "`<JAVA_HOME>\jre\lib\ext`" (where `<JAVA_HOME>` is the JDK installed directory, e.g., "`c:\program files\java\jdk1.8.0_{xx}\jre\lib\ext`").

For Mac

1. Download the latest MySQL JDBC driver from http://www.mysql.com/downloads ⇒ MySQL Connectors ⇒ Connector/J ⇒ Connector/J 5.1.{*xx*} ⇒ select "Platform Independent" ⇒ Compressed TAR Archive (e.g., `mysql-connector-java-5.1.{xx}.tar.gz`, where {*xx*} is the latest release number).
2. Double-click on the downloaded TAR file to expand into folder "`mysql-connector-java-5.1.{xx}`".
3. Open the expanded folder. Copy the JAR file "`mysql-connector-java-5.1.{xx}-bin.jar`" to JDK's extension directory at "`/Library/Java/Extension`".

(For Advanced User Only) You can compile Java database programs without the JDBC driver. But to run the JDBC programs, the JDBC driver's JAR-file must be included in the environment variable CLASSPATH, or the JDK's extension directory, or in the `java`'s command-line option `-cp <paths>`. For example,

```
// For windows
> java -cp .;/path/to/mysql-connector-java-5.1.{xx}-bin.jar JDBCClassToBeRun
// For Macs/Unixes
> java -cp .:/path/to/mysql-connector-java-5.1.{xx}-bin.jar JDBCClassToBeRun
```

2.3 Setup Database

We have to set up a database before embarking on our database programming. We shall call our database "ebookshop" which contains a table called "books", with 5 columns, as below:

```
Database: ebookshop
Table: books
+-------+---------------------------+---------------+---------+-------+
| id    | title                     | author        | price   | qty   |
| (INT) | (VARCHAR(50))             | (VARCHAR(50)) | (FLOAT) | (INT) |
+-------+---------------------------+---------------+---------+-------+
| 1001  | Java for dummies          | Tan Ah Teck   | 11.11   |   11  |
| 1002  | More Java for dummies     | Tan Ah Teck   | 22.22   |   22  |
| 1003  | More Java for more dummies| Mohammad Ali  | 33.33   |   33  |
| 1004  | A Cup of Java             | Kumar         | 44.44   |   44  |
| 1005  | A Teaspoon of Java        | Kevin Jones   | 55.55   |   55  |
+-------+---------------------------+---------------+---------+-------+
```

Start MySQL Server: Start the MySQL server and verify the server's TCP port number from the console messages.

```
// For Windows
> cd {path-to-mysql-bin}   // Check your MySQL installed directory
> mysqld --console

// For Mac OS X
$ cd /usr/local/mysql/bin
$ sudo ./mysqld_safe --console
```

Start a MySQL client: I shall also assume that there is an authorized user called "myuser" with password "xxxx" (otherwise, use "root" user).

```
// For Windows
> cd {path-to-mysql-bin}   // Check your MySQL installed directory
> mysql -u myuser -p

// For Mac OS X
$ cd /usr/local/mysql/bin
$ ./mysql -u myuser -p
```

Run the following SQL statements to create our test database and table.

```
create database if not exists ebookshop;

use ebookshop;

drop table if exists books;
create table books (
   id int,
   title varchar(50),
   author varchar(50),
   price float,
   qty int,
   primary key (id));

insert into books values (1001, 'Java for dummies', 'Tan Ah Teck', 11.11, 11);
insert into books values (1002, 'More Java for dummies', 'Tan Ah Teck', 22.22, 22);
insert into books values (1003, 'More Java for more dummies', 'Mohammad Ali', 33.33, 33);
insert into books values (1004, 'A Cup of Java', 'Kumar', 44.44, 44);
insert into books values (1005, 'A Teaspoon of Java', 'Kevin Jones', 55.55, 55);

select * from books;
```

3. Introduction to JDBC Programming by Examples

A JDBC (Java Database Connectivity) program comprises the following steps:

1. Allocate a `Connection` object, for connecting to the database.
2. Allocate a `Statement` object, under the `Connection` object created.
3. Write a SQL query and execute the query, via the `Statement` and `Connection` created.
4. Process the query result.
5. Close the `Statement` and `Connection` object to free up the resources.

We shall illustrate Java Database programming by the following examples.

3.1 Example 1: SQL SELECT

Try out the following JDBC program (requires JDK 7), which issues an SQL SELECT. Take note that the *source filename* must be the *same* as the *classname*, with extension of "`.java`". Save the program in any directory of your choice (e.g., `d:/myproject`).

> **(For Mac)** Some older Macs do not have JDK 7. Use the JDK 6 example below. You can check your JDK version via command "`javac -version`".

```
1  import java.sql.*;   // Use classes in java.sql package
2
3  // JDK 7 and above
4  public class JdbcSelectTest {  // Save as "JdbcSelectTest.java"
5     public static void main(String[] args) {
6        try (
7           // Step 1: Allocate a database "Connection" object
8           Connection conn = DriverManager.getConnection(
9                 "jdbc:mysql://localhost:8888/ebookshop", "myuser", "xxxx"); // MySQL
10
11          // Step 2: Allocate a "Statement" object in the Connection
12          Statement stmt = conn.createStatement();
13       ) {
14          // Step 3: Execute a SQL SELECT query, the query result
15          //  is returned in a "ResultSet" object.
16          String strSelect = "select title, price, qty from books";
17          System.out.println("The SQL query is: " + strSelect); // Echo For debugging
18          System.out.println();
19
20          ResultSet rset = stmt.executeQuery(strSelect);
21
22          // Step 4: Process the ResultSet by scrolling the cursor forward via next().
23          //  For each row, retrieve the contents of the cells with getXxx(columnName).
24          System.out.println("The records selected are:");
25          int rowCount = 0;
26          while(rset.next()) {   // Move the cursor to the next row
27             String title = rset.getString("title");
28             double price = rset.getDouble("price");
29             int    qty   = rset.getInt("qty");
30             System.out.println(title + ", " + price + ", " + qty);
31             ++rowCount;
32          }
33          System.out.println("Total number of records = " + rowCount);
34
35       } catch(SQLException ex) {
36          ex.printStackTrace();
37       }
38       // Step 5: Close the resources - Done automatically by try-with-resources
39    }
40 }
```

REFERENCES & RESOURCES

1. JDBC Online Tutorial @ http://download.oracle.com/javase/tutorial/jdbc/index.html.
2. JDBC Home Page @ http://java.sun.com/products/jdbc/overview.html.
3. JDBC API Specifications 1.2, 2.1, 3.0, and 4.0 @ http://java.sun.com/products/jdbc.
4. White Fisher, et al., "JDBC API Tutorial and Reference", 3rd eds, Addison Wesley, 2003.
5. MySQL Home Page @ http://www.mysql.org, and documentation.
6. MySQL 5.5 Reference Manual @ http://dev.mysql.com/doc/refman/5.5/en/index.html.
7. SQL.org @ http://www.sql.org.
8. Russell Dyer, "MySQL in a Nutshell", O'Reilly, 2008.

MySQL

https://www3.ntu.edu.sg/home/ehchua/programming/index.html

MySQL by Examples for Beginners

Read "How to Install MySQL and Get Started" on how to install, customize, and get started with MySQL.

1. Summary of MySQL Commands Used in this Tutorial

For detailed syntax, check MySQL manual "SQL Statement Syntax" @ http://dev.mysql.com/doc/refman/5.5/en/sql-syntax.html.

```
-- Database-Level
DROP DATABASE databaseName                  -- Delete the database (irrecoverable!)
DROP DATABASE IF EXISTS databaseName        -- Delete if it exists
CREATE DATABASE databaseName                -- Create a new database
CREATE DATABASE IF NOT EXISTS databaseName -- Create only if it does not exists
SHOW DATABASES                              -- Show all the databases in this server
USE databaseName                            -- Set the default (current) database
SELECT DATABASE()                           -- Show the default database
SHOW CREATE DATABASE databaseName           -- Show the CREATE DATABASE statement

-- Table-Level
DROP TABLE [IF EXISTS] tableName, ...
CREATE TABLE [IF NOT EXISTS] tableName (
   columnName columnType columnAttribute, ...
   PRIMARY KEY(columnName),
   FOREIGN KEY (columnNmae) REFERENCES tableName (columnNmae)
)
SHOW TABLES                 -- Show all the tables in the default database
DESCRIBE|DESC tableName     -- Describe the details for a table
ALTER TABLE tableName ...   -- Modify a table, e.g., ADD COLUMN and DROP COLUMN
ALTER TABLE tableName ADD columnDefinition
ALTER TABLE tableName DROP columnName
ALTER TABLE tableName ADD FOREIGN KEY (columnNmae) REFERENCES tableName (columnNmae)
ALTER TABLE tableName DROP FOREIGN KEY constraintName
SHOW CREATE TABLE tableName         -- Show the CREATE TABLE statement for this tableName

-- Row-Level
INSERT INTO tableName
   VALUES (column1Value, column2Value,...)              -- Insert on all Columns
INSERT INTO tableName
   VALUES (column1Value, column2Value,...), ...         -- Insert multiple rows
INSERT INTO tableName (column1Name, ..., columnNName)
   VALUES (column1Value, ..., columnNValue)             -- Insert on selected Columns
DELETE FROM tableName WHERE criteria
UPDATE tableName SET columnName = expr, ... WHERE criteria
SELECT * | column1Name AS alias1, ..., columnNName AS aliasN
   FROM tableName
   WHERE criteria
   GROUP BY columnName
   ORDER BY columnName ASC|DESC, ...
   HAVING groupConstraints
   LIMIT count | offset count

-- Others
SHOW WARNINGS;   -- Show the warnings of the previous statement
```

2. An Example for the Beginners (But NOT for the dummies)

A MySQL database server contains many databases (or schemas). Each database consists of one or more tables. A table is made up of columns (or fields) and rows (records).

The SQL keywords and commands are NOT case-sensitive. For clarity, they are shown in uppercase. The *names* or *identifiers* (database names, table names, column names, etc.) are case-sensitive in some systems, but not in other systems. Hence, it is best to treat *identifiers* as case-sensitive.

SHOW DATABASES

You can use SHOW DATABASES to list all the existing databases in the server.

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
........
```

The databases "mysql", "information_schema" and "performance_schema" are system databases used internally by MySQL. A "test" database is provided during installation for your testing.

Let us begin with a simple example - a *product sales database*. A product sales database typically consists of many tables, e.g., products, customers, suppliers, orders, payments, employees, among others. Let's call our database "southwind" (inspired from Microsoft's Northwind Trader sample database). We shall begin with the first table called "products" with the following columns (having data types as indicated) and rows:

Database: southwind Table: products				
productID INT	productCode CHAR(3)	name VARCHAR(30)	quantity INT	price DECIMAL(10,2)
1001	PEN	Pen Red	5000	1.23
1002	PEN	Pen Blue	8000	1.25
1003	PEN	Pen Black	2000	1.25
1004	PEC	Pencil 2B	10000	0.48
1005	PEC	Pencil 2H	8000	0.49

2.1 Creating and Deleting a Database - CREATE DATABASE and DROP DATABASE

You can create a new database using SQL command "CREATE DATABASE *databaseName*"; and delete a database using "DROP DATABASE *databaseName*". You could optionally apply condition "IF EXISTS" or "IF NOT EXISTS" to these commands. For example,

```
mysql> CREATE DATABASE southwind;
Query OK, 1 row affected (0.03 sec)

mysql> DROP DATABASE southwind;
Query OK, 0 rows affected (0.11 sec)

mysql> CREATE DATABASE IF NOT EXISTS southwind;
Query OK, 1 row affected (0.01 sec)

mysql> DROP DATABASE IF EXISTS southwind;
Query OK, 0 rows affected (0.00 sec)
```

IMPORTANT: Use SQL DROP (and DELETE) commands with extreme care, as the deleted entities are irrecoverable. **THERE IS NO UNDO!!!**

SHOW CREATE DATABASE

The CREATE DATABASE commands uses some defaults. You can issue a "SHOW CREATE DATABASE *databaseName*" to display the full command and check these default values. We use \G (instead of ';') to display the results vertically. (Try comparing the outputs produced by ';' and \G.)

```
mysql> CREATE DATABASE IF NOT EXISTS southwind;

mysql> SHOW CREATE DATABASE southwind \G
*************************** 1. row ***************************
       Database: southwind
Create Database: CREATE DATABASE `southwind` /*!40100 DEFAULT CHARACTER SET latin1 */
```

2.2 Setting the Default Database - USE

The command "USE *databaseName*" sets a particular database as the default (or current) database. You can reference a table in the default database using *tableName* directly. But you need to use the fully-qualified *databaseName.tableName* to reference a table NOT in the default database.

In our example, we have a database named "southwind" with a table named "products". If we issue "USE southwind" to set southwind as the default database, we can simply call the table as "products". Otherwise, we need to reference the table as "southwind.products".

To display the current default database, issue command "SELECT DATABASE()".

2.3 Creating and Deleting a Table - CREATE TABLE and DROP TABLE

You can create a new table *in the default database* using command "CREATE TABLE *tableName*" and "DROP TABLE *tableName*". You can also apply condition "IF EXISTS" or "IF NOT EXISTS". To create a table, you need to define all its columns, by providing the columns' *name*, *type*, and *attributes*.

Let's create a table "products" in our database "southwind".

```
-- Remove the database "southwind", if it exists.
-- Beware that DROP (and DELETE) actions are irreversible and not recoverable!
mysql> DROP DATABASE IF EXISTS southwind;
Query OK, 1 rows affected (0.31 sec)

-- Create the database "southwind"
mysql> CREATE DATABASE southwind;
Query OK, 1 row affected (0.01 sec)

-- Show all the databases in the server
--   to confirm that "southwind" database has been created.
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| southwind          |
| ......             |

-- Set "southwind" as the default database so as to reference its table directly.
mysql> USE southwind;
Database changed

-- Show the current (default) database
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| southwind  |
+------------+
```

```
-- Show all the tables in the current database.
-- "southwind" has no table (empty set).
mysql> SHOW TABLES;
Empty set (0.00 sec)

-- Create the table "products". Read "explanations" below for the column defintions
mysql> CREATE TABLE IF NOT EXISTS products (
         productID    INT UNSIGNED  NOT NULL AUTO_INCREMENT,
         productCode  CHAR(3)       NOT NULL DEFAULT '',
         name         VARCHAR(30)   NOT NULL DEFAULT '',
         quantity     INT UNSIGNED  NOT NULL DEFAULT 0,
         price        DECIMAL(7,2)  NOT NULL DEFAULT 99999.99,
         PRIMARY KEY  (productID)
       );
Query OK, 0 rows affected (0.08 sec)

-- Show all the tables to confirm that the "products" table has been created
mysql> SHOW TABLES;
+--------------------+
| Tables_in_southwind |
+--------------------+
| products           |
+--------------------+

-- Describe the fields (columns) of the "products" table
mysql> DESCRIBE products;
+-------------+------------------+------+-----+------------+----------------+
| Field       | Type             | Null | Key | Default    | Extra          |
+-------------+------------------+------+-----+------------+----------------+
| productID   | int(10) unsigned | NO   | PRI | NULL       | auto_increment |
| productCode | char(3)          | NO   |     |            |                |
| name        | varchar(30)      | NO   |     |            |                |
| quantity    | int(10) unsigned | NO   |     | 0          |                |
| price       | decimal(7,2)     | NO   |     | 99999.99   |                |
+-------------+------------------+------+-----+------------+----------------+

-- Show the complete CREATE TABLE statement used by MySQL to create this table
mysql> SHOW CREATE TABLE products \G
*************************** 1. row ***************************
       Table: products
Create Table:
CREATE TABLE `products` (
  `productID`    int(10) unsigned  NOT NULL AUTO_INCREMENT,
  `productCode`  char(3)           NOT NULL DEFAULT '',
  `name`         varchar(30)       NOT NULL DEFAULT '',
  `quantity`     int(10) unsigned  NOT NULL DEFAULT '0',
  `price`        decimal(7,2)      NOT NULL DEFAULT '99999.99',
  PRIMARY KEY (`productID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

2.4 Inserting Rows - INSERT INTO

Let's fill up our "products" table with rows. We set the productID of the first record to 1001, and use AUTO_INCREMENT for the rest of records by inserting a NULL, or with a missing column value. Take note that strings must be enclosed with a pair of single quotes (or double quotes).

```
-- Insert a row with all the column values
mysql> INSERT INTO products VALUES (1001, 'PEN', 'Pen Red', 5000, 1.23);
Query OK, 1 row affected (0.04 sec)

-- Insert multiple rows in one command
-- Inserting NULL to the auto_increment column results in max_value + 1
mysql> INSERT INTO products VALUES
         (NULL, 'PEN', 'Pen Blue',  8000, 1.25),
         (NULL, 'PEN', 'Pen Black', 2000, 1.25);
Query OK, 2 rows affected (0.03 sec)
Records: 2  Duplicates: 0  Warnings: 0

-- Insert value to selected columns
-- Missing value for the auto_increment column also results in max_value + 1
mysql> INSERT INTO products (productCode, name, quantity, price) VALUES
         ('PEC', 'Pencil 2B', 10000, 0.48),
         ('PEC', 'Pencil 2H', 8000, 0.49);
Query OK, 2 row affected (0.03 sec)

-- Missing columns get their default values
mysql> INSERT INTO products (productCode, name) VALUES ('PEC', 'Pencil HB');
Query OK, 1 row affected (0.04 sec)

-- 2nd column (productCode) is defined to be NOT NULL
mysql> INSERT INTO products values (NULL, NULL, NULL, NULL, NULL);
ERROR 1048 (23000): Column 'productCode' cannot be null

-- Query the table
mysql> SELECT * FROM products;
+-----------+-------------+-----------+----------+------------+
| productID | productCode | name      | quantity | price      |
+-----------+-------------+-----------+----------+------------+
|      1001 | PEN         | Pen Red   |     5000 |       1.23 |
|      1002 | PEN         | Pen Blue  |     8000 |       1.25 |
|      1003 | PEN         | Pen Black |     2000 |       1.25 |
|      1004 | PEC         | Pencil 2B |    10000 |       0.48 |
|      1005 | PEC         | Pencil 2H |     8000 |       0.49 |
|      1006 | PEC         | Pencil HB |        0 | 9999999.99 |
+-----------+-------------+-----------+----------+------------+
6 rows in set (0.02 sec)

-- Remove the last row
mysql> DELETE FROM products WHERE productID = 1006;
```

2.5 Querying the Database - SELECT

The most common, important and complex task is to query a database for a subset of data that meets your needs - with the SELECT command. The SELECT command has the following syntax:

```
-- List all the rows of the specified columns
SELECT column1Name, column2Name, ... FROM tableName

-- List all the rows of ALL columns, * is a wildcard denoting all columns
SELECT * FROM tableName

-- List rows that meet the specified criteria in WHERE clause
SELECT column1Name, column2Name,... FROM tableName WHERE criteria
SELECT * FROM tableName WHERE criteria
```

For examples,

```
-- List all rows for the specified columns
mysql> SELECT name, price FROM products;
+-----------+-------+
| name      | price |
+-----------+-------+
| Pen Red   |  1.23 |
| Pen Blue  |  1.25 |
| Pen Black |  1.25 |
| Pencil 2B |  0.48 |
| Pencil 2H |  0.49 |
+-----------+-------+
5 rows in set (0.00 sec)

-- List all rows of ALL the columns. The wildcard * denotes ALL columns
mysql> SELECT * FROM products;
+-----------+-------------+-----------+----------+-------+
| productID | productCode | name      | quantity | price |
+-----------+-------------+-----------+----------+-------+
|      1001 | PEN         | Pen Red   |     5000 |  1.23 |
|      1002 | PEN         | Pen Blue  |     8000 |  1.25 |
|      1003 | PEN         | Pen Black |     2000 |  1.25 |
|      1004 | PEC         | Pencil 2B |    10000 |  0.48 |
|      1005 | PEC         | Pencil 2H |     8000 |  0.49 |
+-----------+-------------+-----------+----------+-------+
5 rows in set (0.00 sec)
```

SELECT without Table

You can also issue SELECT without a table. For example, you can SELECT an expression or evaluate a built-in function.

```
mysql> SELECT 1+1;
+-----+
| 1+1 |
+-----+
|   2 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT NOW();
+---------------------+
| NOW()               |
+---------------------+
| 2012-10-24 22:13:29 |
+---------------------+
1 row in set (0.00 sec)

// Multiple columns
mysql> SELECT 1+1, NOW();
+-----+---------------------+
| 1+1 | NOW()               |
+-----+---------------------+
|   2 | 2012-10-24 22:16:34 |
+-----+---------------------+
1 row in set (0.00 sec)
```

2.10 Running a SQL Script

Instead of manually entering each of the SQL statements, you can keep many SQL statements in a text file, called SQL script, and run the script. For example, use a programming text editor to prepare the following script and save as "`load_products.sql`" under "`d:\myProject`" (for Windows) or "`Documents`" (for Mac).

```
DELETE FROM products;
INSERT INTO products VALUES (2001, 'PEC', 'Pencil 3B', 500, 0.52),
                            (NULL, 'PEC', 'Pencil 4B', 200, 0.62),
                            (NULL, 'PEC', 'Pencil 5B', 100, 0.73),
                            (NULL, 'PEC', 'Pencil 6B', 500, 0.47);
SELECT * FROM products;
```

You can run the script either:

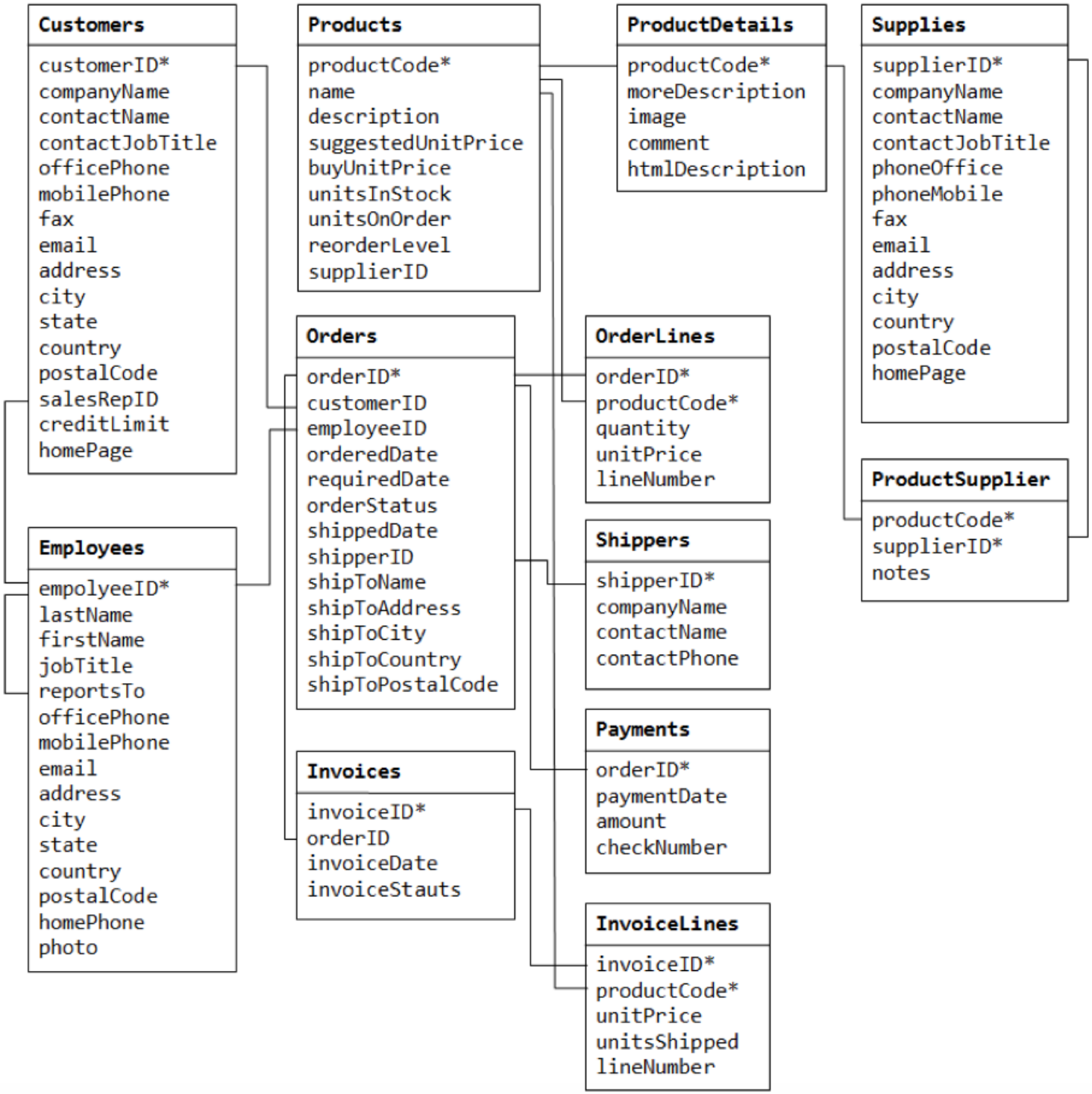
1. via the "`source`" command in a MySQL client. For example, to restore the `southwind` backup earlier:

```
(For Windows)
mysql> source d:/myProject/load_products.sql
   -- Use Unix-style forward slash (/) as directory separator

(For Macs)
mysql> source ~/Documents/load_products.sql
```

2. via the "batch mode" of the `mysql` client program, by re-directing the input from the script:

```
(For Windows)
> cd path-to-mysql-bin
> mysql -u root -p southwind < d:\myProject\load_products.sql

(For Macs)
$ cd /usr/local/mysql/bin
$ ./mysql -u root -p southwind < ~\Documents\load_products.sql
```

7.2 Product Sales Database



Java Servlets

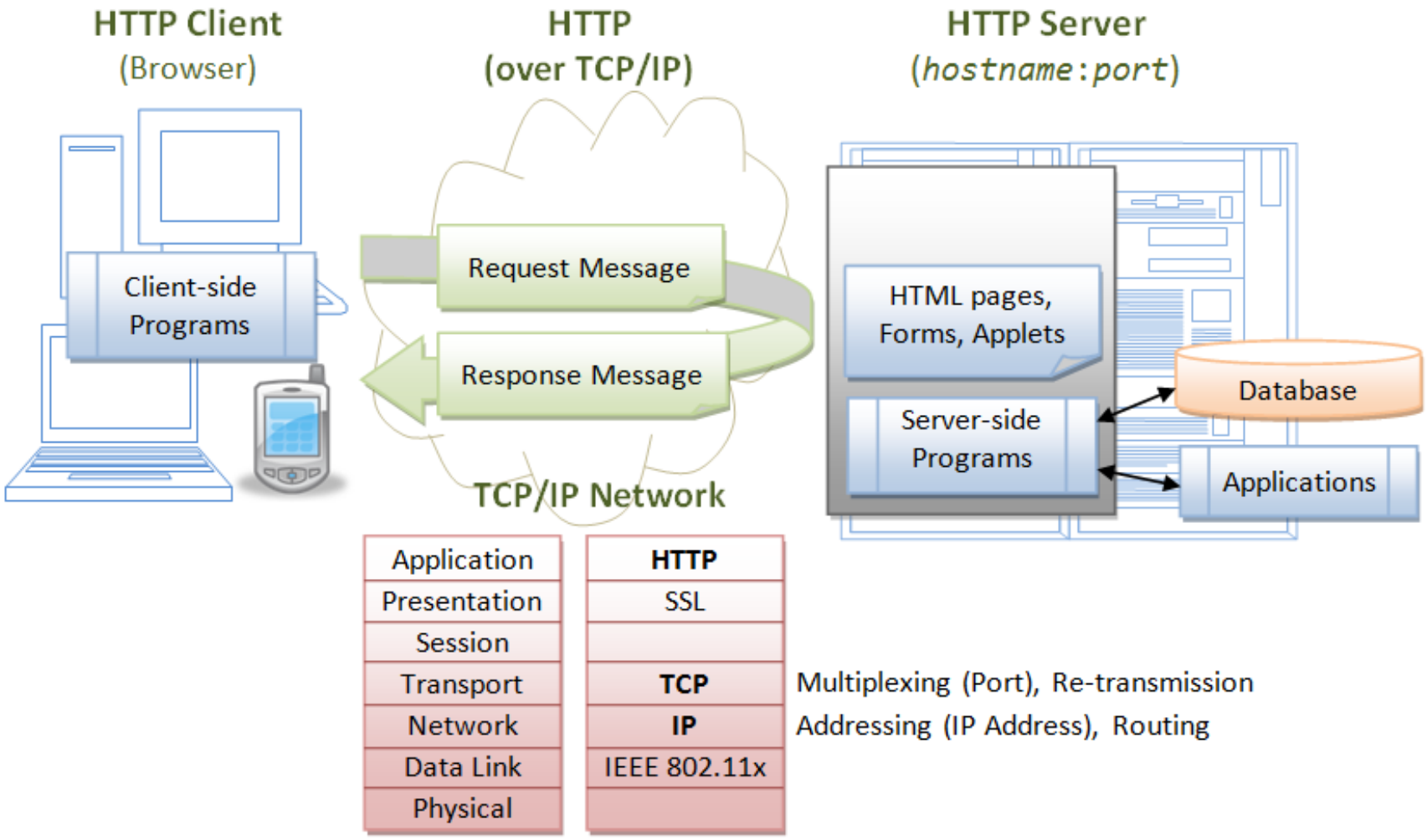
https://www3.ntu.edu.sg/home/ehchua/programming/index.html

Java Server-side Programming

A Java Servlet E-Shop Case Study

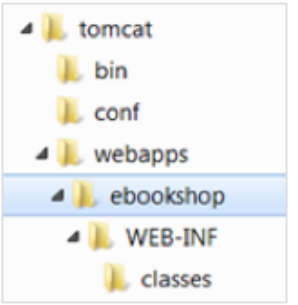
1. Introduction

In this case study, we shall develop an "e-shop" based on the Java Servlet Technology. This e-shop is a typical Internet *business-to-consumer* (B2C) *3-tier client/server database application*, as illustrated below.


HTTP Client
(Browser)
HTTP
(over TCP/IP)
HTTP Server
(hostname:port)
Client-side
Programs
Request Message
Response Message
HTML pages,
Forms, Applets
Server-side
Programs
Database
Applications
TCP/IP Network
Application
Presentation
Session
Transport
Network
Data Link
Physical
HTTP
SSL
TCP
IP
IEEE 802.11x
Multiplexing (Port), Re-transmission
Addressing (IP Address), Routing

4. Setting up the E-Shop Webapp

Step 1: Create the Directory Structure for a new Webapp "ebookshop"



First of all, choose a name for your webapp. In this case study, we shall call it "ebookshop". Navigate to Tomcat's "webapps" directory, and create the following directory structure:

1. Under Tomcat's "webapps" directory, create you web application *root* directory "ebookshop".
2. Under "ebookshop", create a sub-directory "WEB-INF" (case sensitive, uppercase, a dash not underscore).
3. Under "WEB-INF", create a sub-directory "classes" (case sensitive, lowercase, plural).

You need to keep your web resources in the proper directories:

1. "ebookshop": The is called the *context root* (or *document base directory*) of your webapp. You should keep all your HTML files and resources visible to the web users (e.g., CSS, images, scripts) here.
2. "ebookshop\WEB-INF": This directory, although under the context root, is not visible to the web users. This is where you keep your application's configuration files "web.xml".
3. "ebookshop\WEB-INF\classes": This is where you keep all the Java source files and classes.

Step 2: Start the Tomcat Server

To start the Tomcat server:

- For Windows: start a CMD shell and run the batch file "startup.bat" under Tomcat's "bin" directory. Tomcat will be started in a new console window.
- For Mac OS X and Linux: start a Terminal and run "./catalina.sh run"under Tomcat's "bin" directory.

Monitor the Tomcat console, as the information and error messages, and System.out.println() issued by your programs will be sent to this console. Observe the Tomcat's TCP port number.

```
.....
INFO: Initializing ProtocolHandler ["http-bio-9999"]
......
INFO: Deploying web application directory ebookshop
......
INFO: Starting ProtocolHandler ["http-bio-9999"]
......
INFO: Server startup in 699 ms
```

Step 3: Access the Tomcat Server

The Tomcat Server has been started on TCP port 9999. The default TCP port number for HTTP protocol is 80. To access an HTTP server not running on the default TCP port 80, the port number must be explicitly specified in the URL.

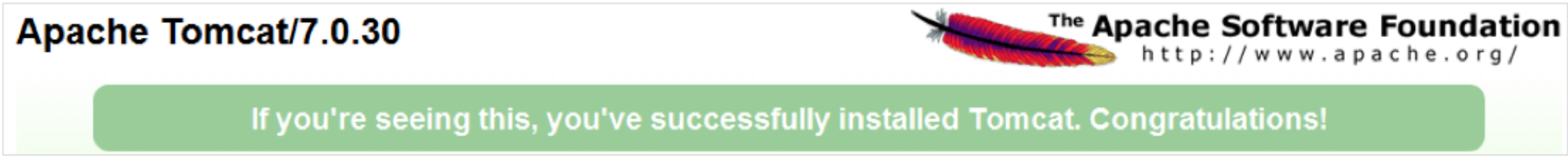
To access your Tomcat Server, start a web browser (e.g., FireFox, IE, Chrome or Safari) and issue the following URL:

```
http://localhost:9999
```

The "localhost" is a special *hostname* (with IP address of 127.0.0.1) meant for *local loop-back* testing.

You could also use the IP address to access your HTTP server. You can find out your IP address by running program such as "`ipconfig`", "`winipcfg`", "`ping`", and etc.

You shall see the *welcome page* of Tomcat Server.



Step 5: Shutting down the Tomcat Server.

To orderly shutdown the Tomcat web server, press *control-c* from the Tomcat's console; or run the script "`shutdown.bat`" (Windows) or "`./shutdown.sh`" (Mac OS/Linux) under Tomcat's "`bin`" directory.

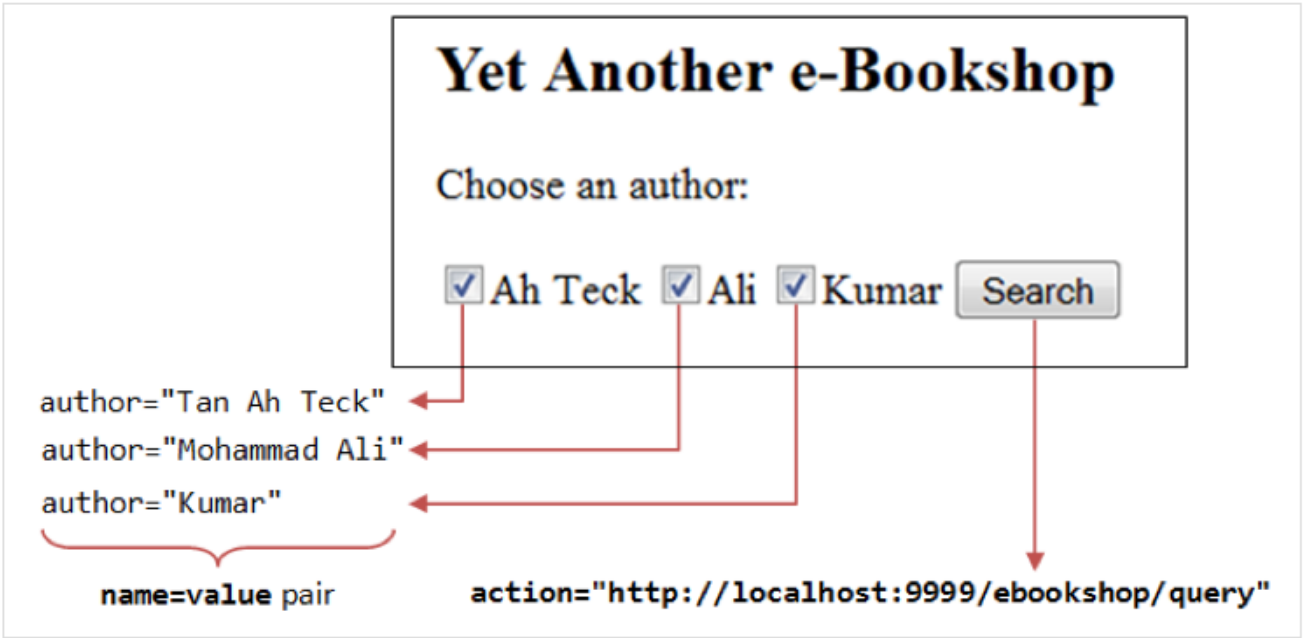
5. Writing a Client-Side HTML Form

Let's write an HTML script to create a query *form* using checkboxes. Save the HTML file as "`querybook.html`" in your application root directory "`ebookshop`".

```
<html>
<head>
  <title>Yet Another e-Bookshop</title>
</head>
<body>
  <h2>Yet Another e-Bookshop</h2>
  <form method="get" action="http://localhost:9999/ebookshop/query">
    Choose an author:<br /><br />
    <input type="checkbox" name="author" value="Tan Ah Teck" />Ah Teck
    <input type="checkbox" name="author" value="Mohammad Ali" />Ali
    <input type="checkbox" name="author" value="Kumar" />Kumar
    <input type="submit" value="Search" />
  </form>
</body>
</html>
```

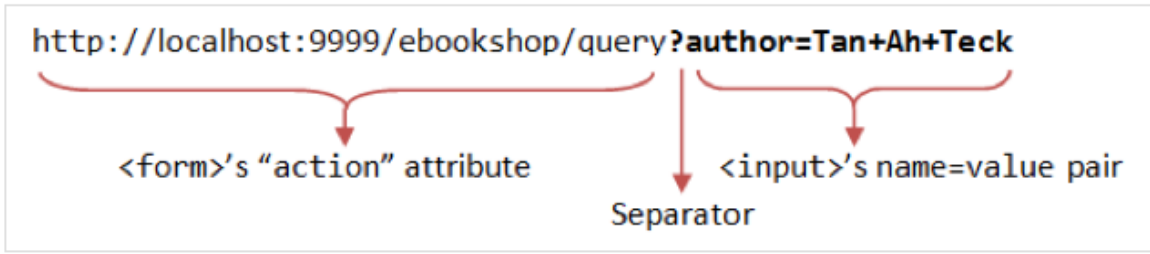
Browse the HTML page by issuing the URL:

```
http://localhost:9999/ebookshop/querybook.html
```



Check a box (e.g., Ah Teck) and click the "Search" button. A request will be issued to the URL specified in the `<form>`'s `action` attribute. You are expected to receive an Error "404 Page Not Found" at this stage as you have yet to write the server-side program (i.e., "`query`").

But observe the URL generated:



The query parameter, in the form of `name=value` pair, are extracted from the `<input>` tag (e.g., `author=Tan+Ah+Tack`). It is appended behind the URL, separated by a `'?'`.

Check two boxes (e.g., "Ah Teck" and "Ali") and submit the request, the URL is:

```
http://localhost:9999/ebookshop/query?author=Tan+Ah+Teck&author=Mohammad+Ali
```

Two `name=value` pairs are sent to the server, separated by an `'&'`.

Also take note that blank is replaced by `'+'`. This is because special characters are not permitted in the URL. They are encoded as *%xx* where *xx* is the hex code in ASCII. For example, `'~'` is encoded as `%7e`; blank is encoded as `%20` or `'+'`.

5. Writing Database Query Servlet

The next step is to write the server-side program, which responses to the client's request by querying the database and returns the query results. We shall use Java servlet technology in our servlet-side programming.

5.1 Java Database Programming

The steps involved in Java database programs are:

1. Allocate a `Connection` object.
2. Allocate a `Statement` object, under the `Connection` object created.
3. Query database:
    a. Execute a SQL `SELECT` query by calling the `executeQuery()` method of the `Statement` object, which returns the query results in a `ResultSet` object; or
    b. Execute a SQL `INSERT|UPDATE|DELETE` command by calling the `executeUpdate()` method of the `Statement` object, which returns an `int` indicating the number of rows affected.
4. Process the query result.
5. Free the resources by closing the `Statement` and `Connection`.

6.2 Database Servlet

Let write a servlet that queries the database based on the client's request.

Step 1: Write the Servlet "QueryServlet.java"

Enter the following codes and save as "QueryServlet.java" under your web application "classes" directory, i.e., "**ebookshop\WEB-INF\classes**". **You must keep all your servlets in "ebookshop\WEB-INF\classes**", because that is where Tomcat picks up the servlets.

```
1  // Saved as "ebookshop\WEB-INF\classes\QueryServlet.java".
2  import java.io.*;
3  import java.sql.*;
4  import javax.servlet.*;
5  import javax.servlet.http.*;
6
7  public class QueryServlet extends HttpServlet {  // JDK 6 and above only
8
9     // The doGet() runs once per HTTP GET request to this servlet.
10    @Override
11    public void doGet(HttpServletRequest request, HttpServletResponse response)
12                      throws ServletException, IOException {
13       // Set the MIME type for the response message
14       response.setContentType("text/html");
15       // Get a output writer to write the response message into the network socket
16       PrintWriter out = response.getWriter();
17
18       Connection conn = null;
19       Statement stmt = null;
20       try {
21          // Step 1: Create a database "Connection" object
22          // For MySQL
23          conn = DriverManager.getConnection(
24             "jdbc:mysql://localhost:8888/ebookshop", "myuser", "xxxx");  // <<== Check
25          // For MS Access
26          // conn = DriverManager.getConnection("jdbc:odbc:ebookshopODBC");
27
28          // Step 2: Create a "Statement" object inside the "Connection"
29          stmt = conn.createStatement();
30
31          // Step 3: Execute a SQL SELECT query
32          String sqlStr = "SELECT * FROM books WHERE author = "
33                + "'" + request.getParameter("author") + "'"
34                + " AND qty > 0 ORDER BY author ASC, title ASC";
35
36          // Print an HTML page as output of query
37          out.println("<html><head><title>Query Results</title></head><body>");
38          out.println("<h2>Thank you for your query.</h2>");
39          out.println("<p>You query is: " + sqlStr + "</p>"); // Echo for debugging
40          ResultSet rset = stmt.executeQuery(sqlStr); // Send the query to the server
```

Step 3: Configure the Servlet

Configure the servlet by creating a configuration file called "web.xml", and save it under the "ebookshop\WEB-INF" directory.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app version="3.0"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    metadata-complete="true">

  <!-- To save as "ebookshop\WEB-INF\web.xml" -->

  <servlet>
    <servlet-name>EBookShopQuery</servlet-name>
    <servlet-class>QueryServlet</servlet-class>
  </servlet>

  <!-- Note: All <servlet> elements must be placed
       in front of <servlet-mapping> elements -->

  <servlet-mapping>
    <servlet-name>EBookShopQuery</servlet-name>
    <url-pattern>/query</url-pattern>
  </servlet-mapping>

</web-app>
```

In the above configuration, we map "QueryServlet.class" to request URL "/query" (thru an arbitrary but unique <servlet-name> called "EBookShopQuery"), under this webapp "ebookshop". In other words, the full request URL for this servlet is http://*hostname*:*port*/**ebookshop/query**. This is exactly what we have written in our HTML <form>'s action attribute.

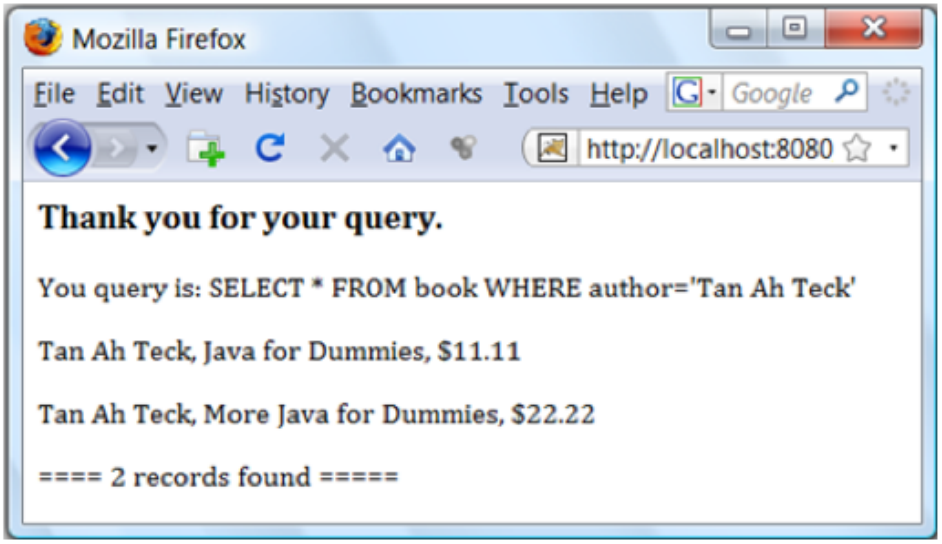
(For Advanced Users) For Tomcat 7, which supports Servlet 3.0, you can use the @WebServlet annotation in the "QueryServlet.java" to deploy the servlet, instead of writing the "web.xml", as follows:

```
@WebServlet("/query")
public class QueryServlet extends HttpServlet { ..... }
```

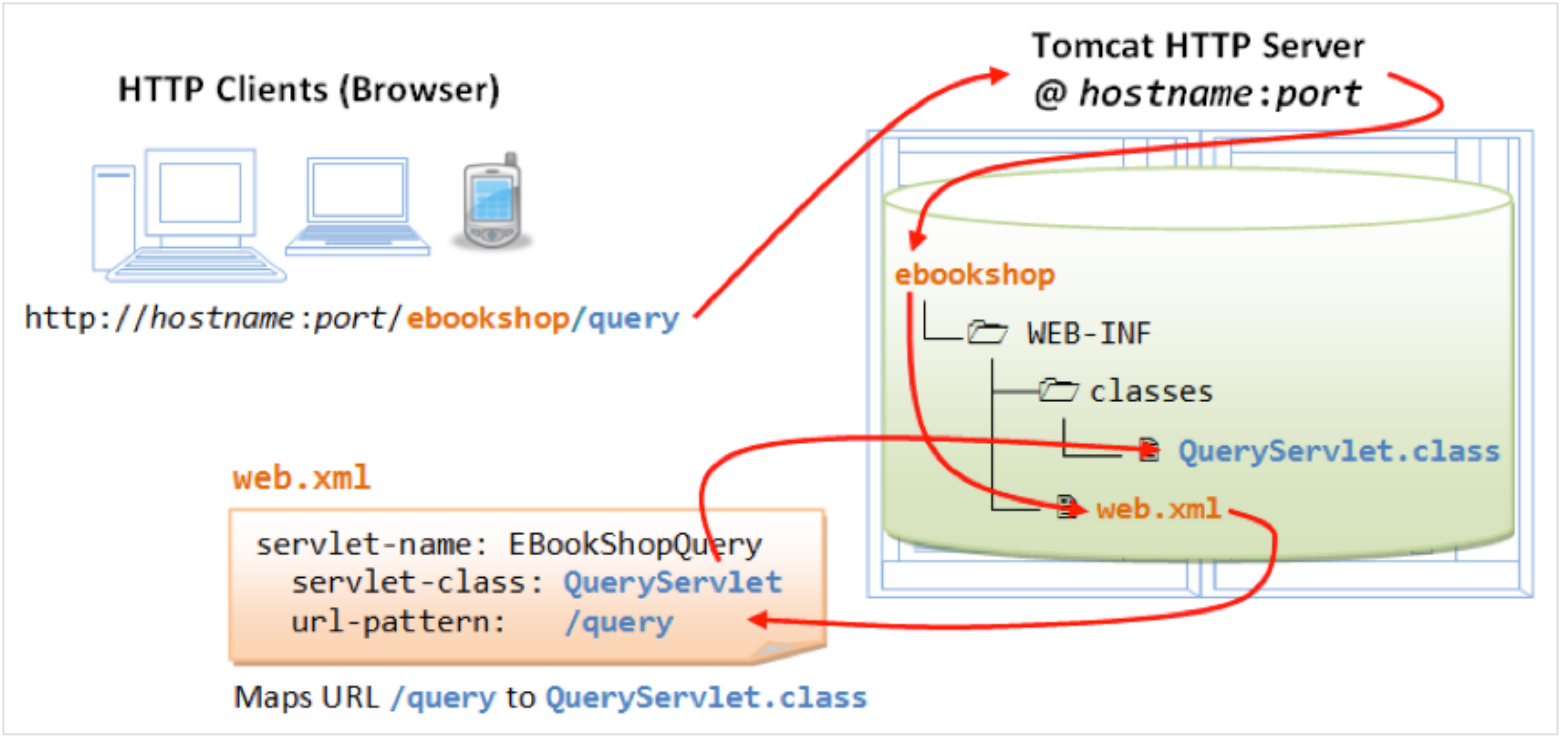
Step 4: Test the Servlet

You can now try to invoke the servlet by issuing a request URL with proper query parameter:

```
http://localhost:9999/ebookshop/query?author=Tan+Ah+Teck
```



The request URL "`/query`" is mapped to the servlet "`QueryServlet.class`", as configured in the application "`web.xml`", as follows:



Tutorial Continues…

REFERENCES & RESOURCES

1. JDK (aka Java SE) Home Page @ http://java.sun.com/javase (or http://www.oracle.com/technetwork/java/javase/overview/index.html).
2. Apache Tomcat Home Page @ http://tomcat.apache.org.
3. Java Database Connectivity (JDBC) Home Page @ http://java.sun.com/products/jdbc.
4. Java Servlets Home Page @ http://java.sun.com/products/servlet. Servlet Developers @ http://java.net/projects/servlet/.
5. The Java EE 6 Tutorial, Chapter 10 "*Java Servlet Technology*", December 2009 @ http://java.sun.com/javaee/6/docs/tutorial/doc/bnafd.html.
6. The Java EE 5 Tutorial, Chapter 4 "*Java Servlet Technology*", October 2008 @ http://java.sun.com/javaee/5/docs/tutorial/doc/bnafd.html.
7. The J2EE 1.4 Tutorial, Chapter 11 "*Java Servlet Technology*", December, 2005 @ http://java.sun.com/j2ee/1.4/docs/tutorial/doc/.
8. The J2EE 1.3 Tutorial "*Java Servlet Technology*" @ http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html.
9. White Fisher, et al., "*JDBC API Tutorial and Reference*", 3rd eds, Addison Wesley.
10. MySQL Mother Site @ www.mysql.com, and MySQL 5.5 Reference Manual @ http://dev.mysql.com/doc/refman/5.5/en/index.html.
11. SQL.org @ http://www.sql.org.
12. Marty Hall, et al., "*Core Servlets and JavaServer Pages*", vol.1 (2nd eds, 2003) and vol. 2 (2nd eds, 2006), Prentice Hall.
13. RFC2616 "*Hypertext Transfer Protocol HTTP 1.1*", World-Wide-Web Consortium (W3C), June 1999.
14. "*HTML 4.01 Specification*", W3C Recommendation, 24 Dec 1999.
15. Eric Ladd and Jim O'Donnell, "*Using HTML, Java and CGI*" and "*Using HTML 4, XML and Java 1.2*", Que.

Java Server Pages

https://www3.ntu.edu.sg/home/ehchua/programming/index.html

Java Server-side Programming

Getting started with JSP by Examples

1. Introduction

JavaServer Page (JSP) is Java's answer to the popular Microsoft's *Active Server Pages* (ASP). JSP, like ASP, provides a simplified and fast mean to generate *dynamic* web contents. It allows you to mix *static* HTML with *dynamically generated* HTML - in the way that the *business logic* and the *presentation* are well separated.

The advantages of JSP are:

1. **Separation of static and dynamic contents**: JSP enables the separation of *static* contents from *dynamic* contents. The dynamic contents are generated via programming logic and inserted into the *static template*. This greatly simplifies the creation and maintenance of web contents.
2. **Reuse of components and tag libraries:** The dynamic contents can be provided by reusable components such as JavaBean, Enterprise JavaBean (EJB) and tag libraries - you do not have to re-inventing the wheels.
3. **Java's power and portability**

JSPs are Internally Compiled into Java Servlets

That is to say, anything that can be done using JSPs can also be accomplished using Java servlets. However, it is important to note that servlets and JSPs are *complementary* technologies, NOT replacement of each other. Servlet can be viewed as "***HTML inside Java***", which is better for implementing business logic - as it is Java dominant. JSP, on the other hand, is "***Java inside HTML***", which is superior for creating presentation - as it is HTML dominant. In a typical *Model-View-Control* (MVC) application, servlets are often used for the Controller (C), which involves complex programming logic. JSPs are often used for the View (V), which mainly deals with presentation. The Model (M) is usually implemented using JavaBean or EJB.

2. First JSP Example - "*Java inside HTML*"

Let's begin with a simple JSP example. We shall use the webapp called "`hello`" that we have created in our earlier exercise. Use a programming text editor to enter the following HTML/JSP codes and save as "`first.jsp`" (the file type of "`.jsp`" is mandatory) in your webapp (web context) home directory (i.e., "`webapps\hello`".

```
<html>
<head><title>First JSP</title></head>
<body>
  <%
    double num = Math.random();
    if (num > 0.95) {
  %>
      <h2>You'll have a luck day!</h2><p>(<%= num %>)</p>
  <%
    } else {
  %>
      <h2>Well, life goes on ... </h2><p>(<%= num %>)</p>
  <%
    }
  %>
  <a href="<%= request.getRequestURI() %>"><h3>Try Again</h3></a>
</body>
</html>
```

To execute the JSP script: Simply start your Tomcat server and use a browser to issue an URL to browse the JSP page (i.e., `http://localhost:8080/hello/first.jsp`).

From your browser, choose the "View Source" option to check the response message. It should be either of the followings depending on the random number generated.

```
<html>
<h2>You'll have a luck day!</h2>
<p>(0.987)</p>
<a href="first.jsp"><h3>Try Again</h3></a>
</html>
```

```
<html>
<h2> Well, life goes on ... </h2>
<p>(0.501)</p>
<a href="first.jsp"><h3>Try Again</h3></a>
</html>
```

It is important to note that the client is not able to "view" the original JSP script (otherwise, you may have security exposure), but merely the result generated by the script.

4. Second JSP example - Echoing HTML Request Parameters

Enter the following JSP script and save as "`echo.jsp`" in your *webapp's root directory*.

```
<html>
<head>
  <title>Echoing HTML Request Parameters</title>
</head>
<body>
  <h3>Choose an author:</h3>
  <form method="get">
    <input type="checkbox" name="author" value="Tan Ah Teck">Tan
    <input type="checkbox" name="author" value="Mohd Ali">Ali
    <input type="checkbox" name="author" value="Kumar">Kumar
    <input type="submit" value="Query">
  </form>

  <%
  String[] authors = request.getParameterValues("author");
  if (authors != null) {
  %>
    <h3>You have selected author(s):</h3>
    <ul>
  <%
      for (int i = 0; i < authors.length; ++i) {
  %>
        <li><%= authors[i] %></li>
  <%
      }
  %>
    </ul>
    <a href="<%= request.getRequestURI() %>">BACK</a>
  <%
  }
  %>
</body>
</html>
```

Browse the JSP page created and study the generated servlet.

5. JSP Scripting Elements

JSP provides the following scripting elements:

- JSP Comment `<%-- comments -->`
- JSP Expression `<%= Java Expression %>`
- JSP Scriptlet `<% Java Statement(s) %>`
- JSP Directive `<%@ page|include ... %>`

To simplify the access of the HTTP *request* and *response* messages, JSP has *pre-defined* the following variables:

- `request`: corresponds to the HTTP request message.
- `response`: corresponds to the HTTP response message.
- `out`: corresponds to the HTTP response message's output stream.
- others such as `session`, `page`, `application`, `pageContext`, which are outside the scope of this tutorial.

5.1 JSP comment `<%-- comments --%>`

JSP comments `<%-- comments --%>` are ignored by the JSP engine. For example,

```
<%-- anything but a closing tag here will be ignored -->
```

Note that HTML comment is `<!-- comments -->`. JSP expression within the HTML comment will be evaluated. For example,

```
<!-- HTML comments here <%=  Math.random() %> more comments -->
```

5.2 JSP Expression <%= Java Expression %>

JSP Expression can be used to insert a *single* Java expression directly into the response message. This expression will be placed inside a `out.print()` method. Hence, the expression will be evaluated and printed out as part of the response message. Any valid Java expression can be used. There is no semi-colon at the end of the expression. For examples:

```
<p>The square root of 5 is <%= Math.sqrt(5) %></p>
<h5><%= item[10] %></h5>
<p>Current time is: <%=  new java.util.Date() %></p>
```

The above JSP expressions will be converted to:

```
out.write("<p>The square root of 5 is ");
out.print( Math.sqrt(5) );
out.write("</p>");
out.write("<h5>");
out.print( item[10] );
out.write("</h5>");
out.write("<p>Current  time is: ");
out.print( new java.util.Date()  );
out.write("</p>");
```

You can use the pre-defined variables, such as `request`, in the expressions. For examples:

```
<p>You have choose author <%= request.getParameter("author") %></p>
<%= request.getRequestURI() %>
<%= request.getHeader("Host") %>
```

5.3 JSP Scriptlet <% Java statement(s) %>

JSP scriptlets allow you to do more *complex operations* than inserting a single Java expression (with the JSP expression). JSP scriptlets let you insert an arbitrary sequence of valid Java statement(s) into the `service()` method of the converted servlet. All the Java statements in a scriptlet are to be terminated with a semi-colon. For example:

```
<%
  String author = request.getParameter("author");
  if (author != null && !author.equals(""))) {
%>
    <p>You have choose author <%= author %></p>
<%
  }
%>
```

In the converted servlet, the above will be inserted into the `service()` method as follows:

```
String author = request.getParameter("author");
if (author != null && !author.equals(""))) {
  out.write("<p>You have choose author ");
  out.print( author );
  out.write("</p>");
}
```

Notice that the Java codes inside a scriptlet are inserted exactly as they are written, and used as the programming logic. The HTML codes are passed to an `out.write()` method and written out as part of the response message.

5.4 JSP Directive <%@ page|include ... %>

JSP directives provide instructions to the JSP engine. The syntax of the JSP directive is:

```
<%@ directive_name
  attribute1="value1"
  attribute2="value2"
  ......
  attributeN="valueN" %>
```

JSP page Directive

The "page" directive lets you import classes and customize the page properties. For examples,

```
<%-- import package java.sql.*  -->
<%@ page import="java.sql.*" %>

<%-- Set the output MIME type  -->
<%@ page contentType="image/gif" %>

<%-- Set an information message for getServletInfo() method -->
<%@ page info="Hello-world example" %>
```

JSP include Directive

The "include" directive lets you include another file(s) at the time when the JSP page is compiled into a servlet. You can include any JSP files, or static HTML files. You can use include directive to include navigation bar, copyright statement, logo, etc. on every JSP pages. The syntax is:

```
<%@ include file="url" %>
```

For example:

```
<%@ include file="header.html" %>
......
<%@ include file="footer.html" %>
```

6. JSP Database Example

Let's revisit our e-shop, which was created using Java Servlet.

Database

```
Database: ebookshop
Table: books
+-------+---------------------------+---------------+---------+-------+
| id    | title                     | author        | price   | qty   |
| (INT) | (VARCHAR(50))             | (VARCHAR(50)) | (FLOAT) | (INT) |
+-------+---------------------------+---------------+---------+-------+
| 1001  | Java for dummies          | Tan Ah Teck   | 11.11   | 11    |
| 1002  | More Java for dummies     | Tan Ah Teck   | 22.22   | 22    |
| 1003  | More Java for more dummies| Mohammad Ali  | 33.33   | 33    |
| 1004  | A Cup of Java             | Kumar         | 44.44   | 44    |
| 1005  | A Teaspoon of Java        | Kevin Jones   | 55.55   | 55    |
+-------+---------------------------+---------------+---------+-------+
```

Querying - "query.jsp"

Let's create the query page called "query.jsp".

```
<html>
<head>
  <title>Book Query</title>
</head>
<body>
  <h1>Another E-Bookstore</h1>
  <h3>Choose Author(s):</h3>
  <form method="get">
    <input type="checkbox" name="author" value="Tan Ah Teck">Tan
    <input type="checkbox" name="author" value="Mohd Ali">Ali
    <input type="checkbox" name="author" value="Kumar">Kumar
    <input type="submit" value="Query">
  </form>

  <%
    String[] authors = request.getParameterValues("author");
    if (authors != null) {
  %>
  <%@ page import = "java.sql.*" %>
  <%
      Connection conn = DriverManager.getConnection(
          "jdbc:mysql://localhost:8888/ebookshop", "myuser", "xxxx"); // <== Check!
      // Connection conn =
      //    DriverManager.getConnection("jdbc:odbc:eshopODBC");  // Access
      Statement stmt = conn.createStatement();

      String sqlStr = "SELECT * FROM books WHERE author IN (";
      sqlStr += "'" + authors[0] + "'";  // First author
      for (int i = 1; i < authors.length; ++i) {
         sqlStr += ", '" + authors[i] + "'";  // Subsequent authors need a leading commas
      }
      sqlStr += ") AND qty > 0 ORDER BY author ASC, title ASC";

      // for debugging
      System.out.println("Query statement is " + sqlStr);
      ResultSet rset = stmt.executeQuery(sqlStr);
  %>
      <hr>
      <form method="get" action="order.jsp">
        <table border=1 cellpadding=5>
          <tr>
            <th>Order</th>
            <th>Author</th>
```

Ordering - "order.jsp"

Let's write the "order.jsp" for processing the order, by updating the appropriate records in the database.

```
<html>
<head>
  <title>Order Book</title>
</head>

<body>
  <h1>Another E-Bookstore</h1>
  <h2>Thank you for ordering...</h2>

  <%
    String[] ids = request.getParameterValues("id");
    if (ids != null) {
  %>
  <%@ page import = "java.sql.*" %>
  <%
      Connection conn = DriverManager.getConnection(
          "jdbc:mysql://localhost:8888/ebookshop", "myuser", "xxxx"); // <== Check!
      // Connection conn =
      //    DriverManager.getConnection("jdbc:odbc:eshopODBC");  // Access
      Statement stmt = conn.createStatement();
      String sqlStr;
      int recordUpdated;
      ResultSet rset;
  %>
      <table border=1 cellpadding=3 cellspacing=0>
        <tr>
          <th>Author</th>
          <th>Title</th>
          <th>Price</th>
          <th>Qty In Stock</th>
        </tr>
  <%
      for (int i = 0; i < ids.length; ++i) {
        // Subtract the QtyAvailable by one
        sqlStr = "UPDATE books SET qty = qty - 1 WHERE id = " + ids[i];
        recordUpdated = stmt.executeUpdate(sqlStr);
        // carry out a query to confirm
        sqlStr = "SELECT * FROM books WHERE id =" + ids[i];
        rset = stmt.executeQuery(sqlStr);
        while (rset.next()) {
  %>
          <tr>
            <td><%= rset.getString("author") %></td>
```

REFERENCES & RESOURCES

1. JavaServer Pages Technology @ http://java.sun.com/products/jsp.
2. Java Servlet Technology @ http://java.sun.com/products/servlet.
3. Apache Tomcat @ http://tomcat.apache.org, Apache Software Foundation.
4. The Java EE 5 Tutorial @ http://java.sun.com/javaee/5/docs/tutorial/doc/.
5. Marty Hall, et al., "*Core Servlets and JavaServer Pages*", vol.1 (2nd eds, 2003) and vol. 2 (2nd eds, 2006), Prentice Hall.
6. Java Database Connectivity (JDBC) @ http://java.sun.com/products/jdbc.
7. RFC2616 "*Hypertext Transfer Protocol HTTP 1.1*", World-Wide-Web Consortium (W3C), June 1999.
8. "*HTML 4.01 Specification*", W3C Recommendation, 24 Dec 1999.

Java SE Security

To be Continued…

Next: Web Technology Lab & Java Readings/Videos

http://docs.oracle.com/javase/tutorial/security/index.html

See Also…

Web Developer Skills

Learn to build professional websites and applications as used by real businesses.

Make a Website

97%

Make an Interactive Website

Build the Flipboard home page and learn how to add interactivity to your website.

Make a Rails App

Build the Etsy site using Ruby on Rails and learn the essentials of database-backed web applications.

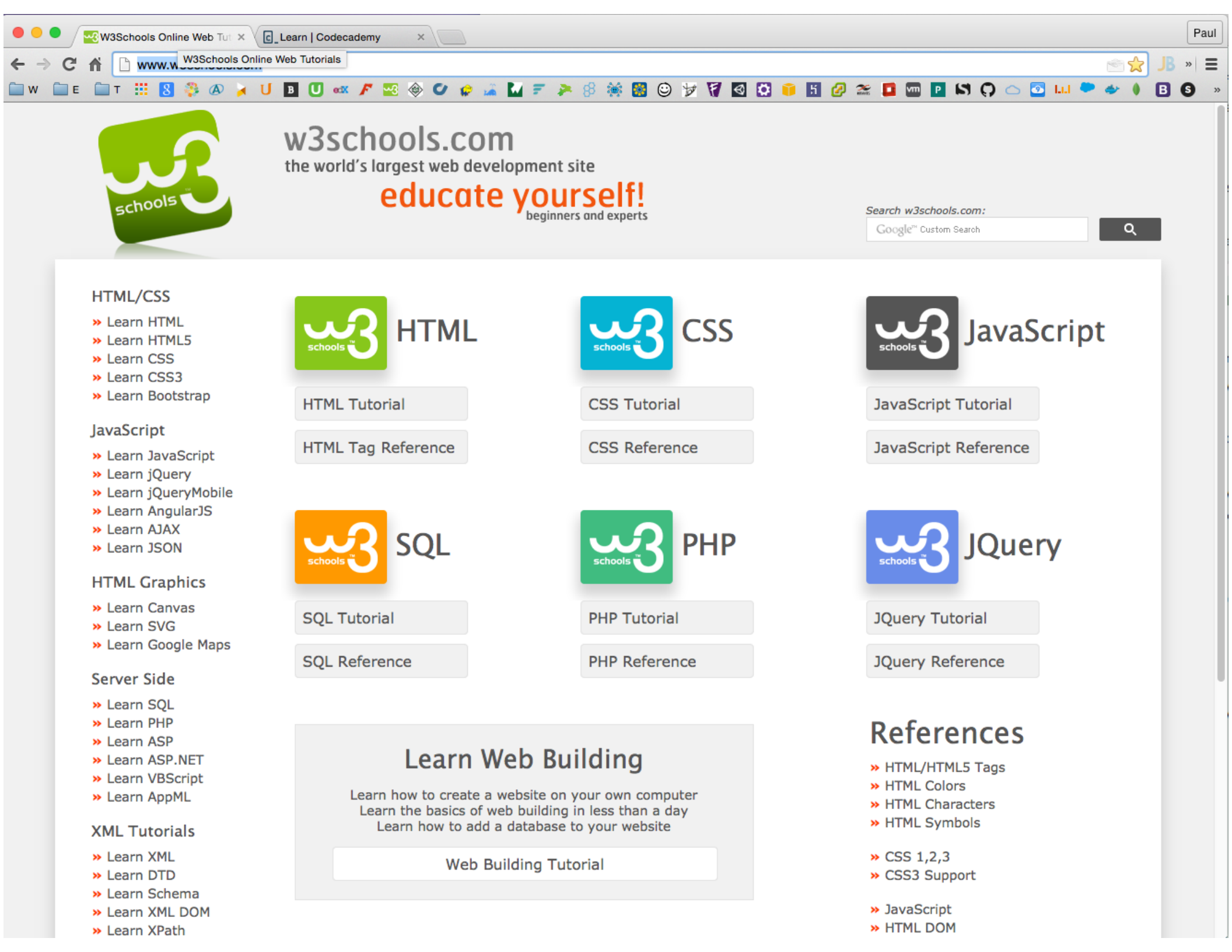
Language Skills

Learn core programming concepts and syntax for the world's most popular languages.

HTML & CSS

JavaScript

jQuery


W3Schools Online Web Tut
Learn | Codecademy
Paul
W3Schools Online Web Tutorials
www.w3schools.com
w3schools.com
the world's largest web development site
educate yourself!
beginners and experts
Search w3schools.com:
Google™ Custom Search
HTML/CSS
» Learn HTML
» Learn HTML5
» Learn CSS
» Learn CSS3
» Learn Bootstrap
JavaScript
» Learn JavaScript
» Learn jQuery
» Learn jQueryMobile
» Learn AngularJS
» Learn AJAX
» Learn JSON
HTML Graphics
» Learn Canvas
» Learn SVG
» Learn Google Maps
Server Side
» Learn SQL
» Learn PHP
» Learn ASP
» Learn ASP.NET
» Learn VBScript
» Learn AppML
XML Tutorials
» Learn XML
» Learn DTD
» Learn Schema
» Learn XML DOM
» Learn XPath
HTML
HTML Tutorial
HTML Tag Reference
CSS
CSS Tutorial
CSS Reference
JavaScript
JavaScript Tutorial
JavaScript Reference
SQL
SQL Tutorial
SQL Reference
PHP
PHP Tutorial
PHP Reference
JQuery
JQuery Tutorial
JQuery Reference
Learn Web Building
Learn how to create a website on your own computer
Learn the basics of web building in less than a day
Learn how to add a database to your website
Web Building Tutorial
References
» HTML/HTML5 Tags
» HTML Colors
» HTML Characters
» HTML Symbols
» CSS 1,2,3
» CSS3 Support
» JavaScript
» HTML DOM

Java Tutorials

Java and Related Technologies

Apache ANT

Apache POI PPT

Apache POI Word

Apache POI

AWT

Design Patterns

EasyMock

Eclipse

EJB

Guava

Hibernate

iBATIS

JACKSON

Jasper Reports

Java XML

JAVA Programming

Java.io package

JAVA Programming

Java.lang package

Java.math

Java.util package

JAVA-8

Java Programming Examples