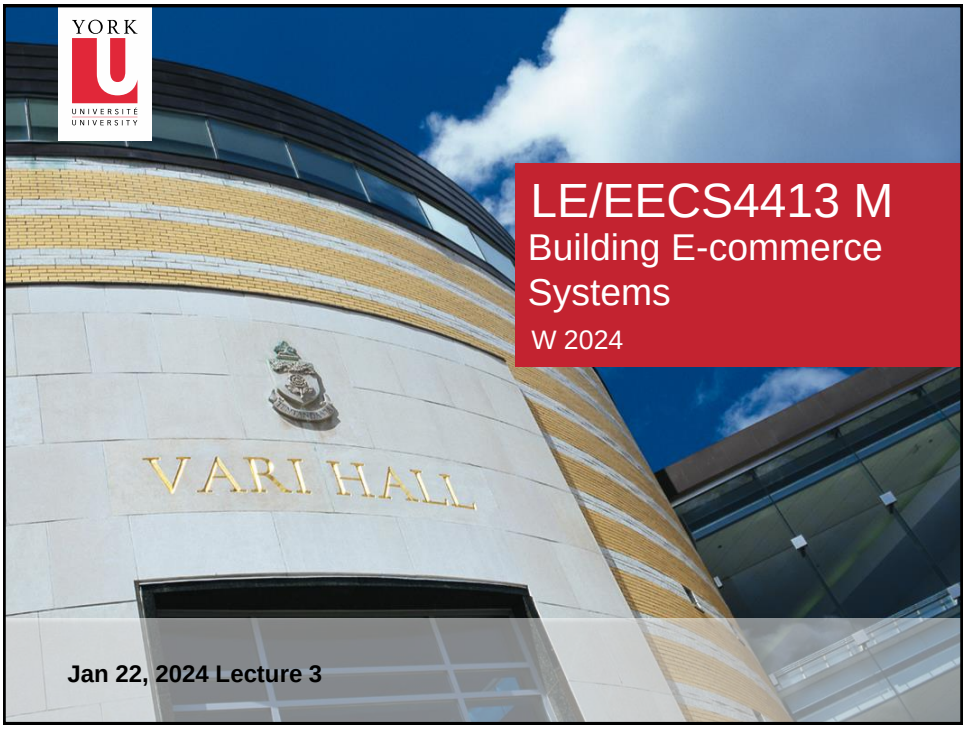



# LE/EECS4413 M
## Building E-commerce Systems
W 2024

**Jan 22, 2024 Lecture 3**

1

# Acknowledgements

Some of the covered materials is based on the previous EECS4413 offerings from:

- *Hamzeh Roumani*
- *Vincent Chu*
- *Marin Litiou*
- *Alvine Belle*
- *Kostas Kontogiannis.*

2

2

# Main topics (tentative)

- Web App Architecture. Preliminary knowledge/Review
  - Client side: HTML CSS JavaScript
  - UML, design patterns, Java (cmd, thread, serialization),
- Client-Server, low level: socket programming
- Web applications (server side)
  - LAMP/CGI
  - Java Servlet
  - JSP, JavaBean, MVC pattern
  - SQL, Database access: JDBC. JPA
  - More: listener, filter, Ajax, JSON

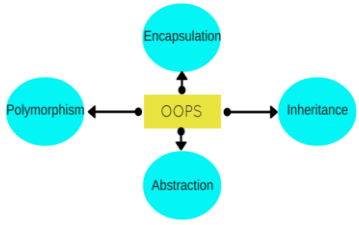

- Web (RESTful) services, micro services
- Advanced topics (TBD): Deployments: Docker container, Node JS, React, Angular, Spring
- Other advanced topics (TBD) More design patterns, Performance, security





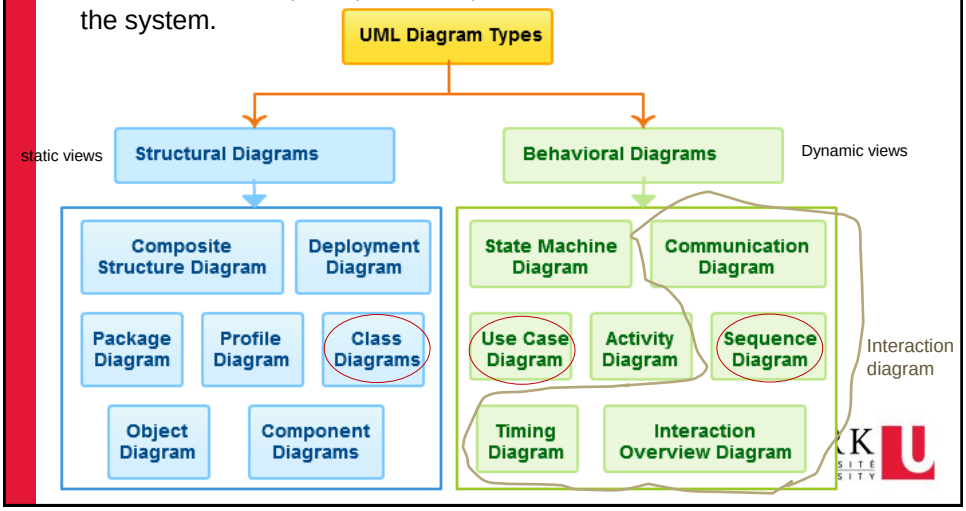
# Preliminary knowledge

- Client side: HTML CSS JS
- Java review
  - OOP, UML, design pattern
  - Command line, class files, jar files
  - Multithreading in JAVA
  - Serialization
- Relational database and SQL



# UML diagrams

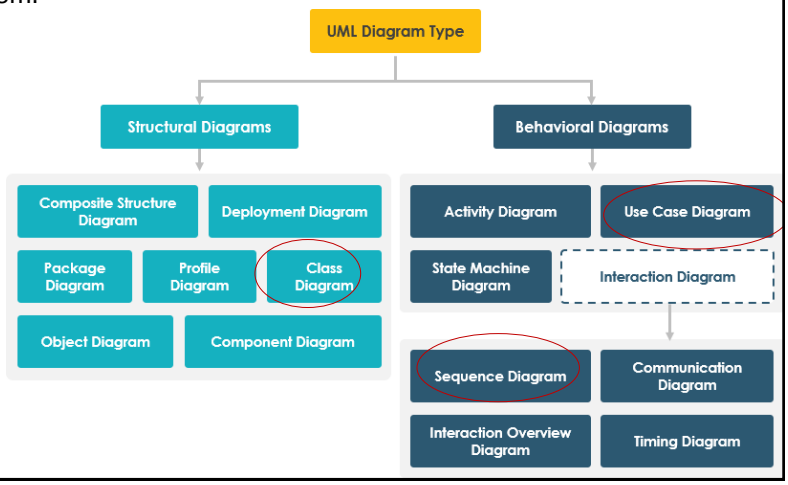
- A UML diagram is a diagram based on the UML (Unified Modeling Language) with the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system.




7

# UML diagrams

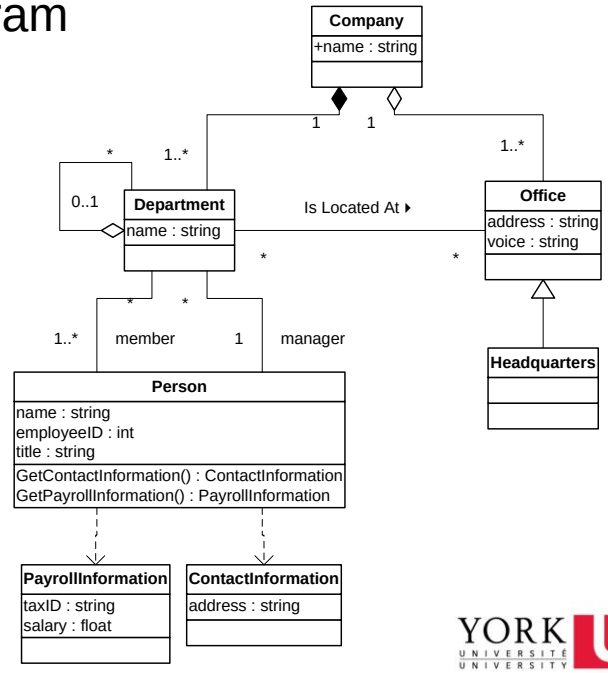
- A UML diagram is a diagram based on the UML (Unified Modeling Language) with the purpose of visually representing a system along with its main actors, roles, actions, artifacts or classes, in order to better understand, alter, maintain, or document information about the system.




8

8

# Class Diagram Example

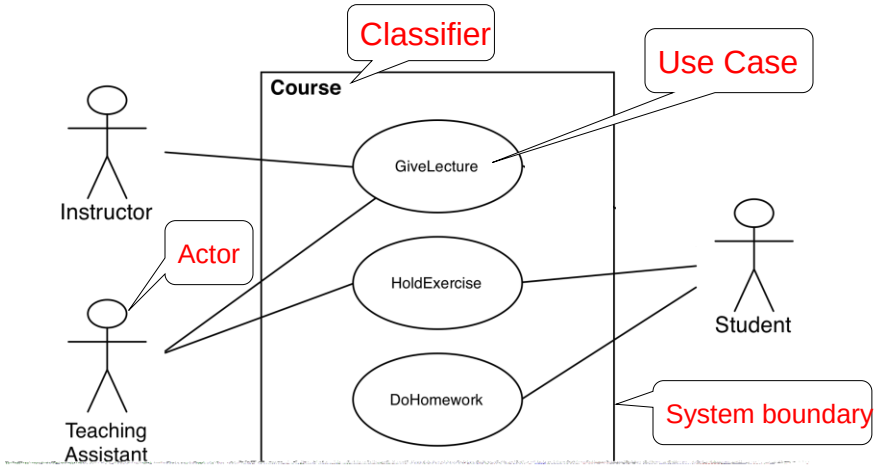



What are some things that are <u>not</u> represented in a UML class diagram?

- details of how the classes interact with each other
- algorithmic details; how a particular behavior is implemented

9

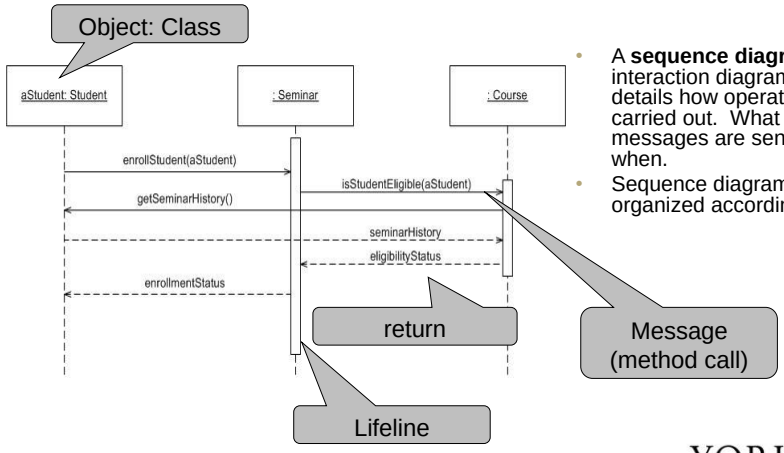
# Use case diagrams




Use case diagrams represent the functionality of the system from user's point of view

10

# Sequence Diagram

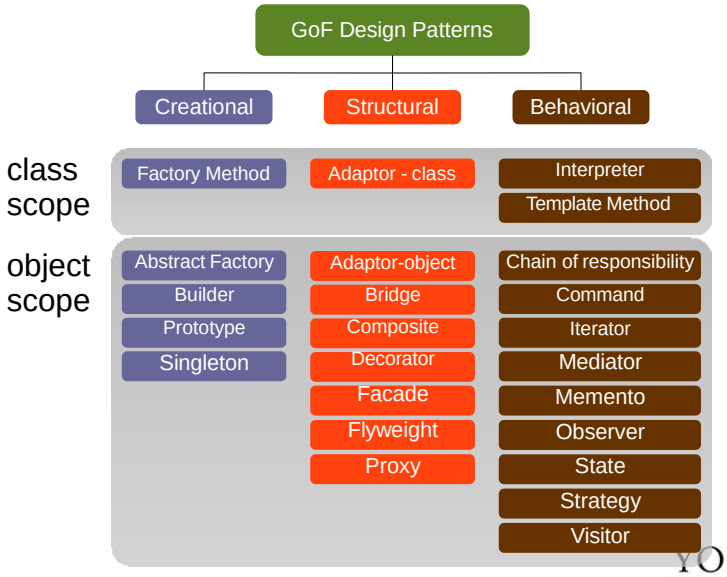



- A **sequence diagram** is an interaction diagram that details how operations are carried out. What messages are sent and when.
- Sequence diagrams are organized according to time

11

11

# Design Patterns Classification




12

12

# Java review

Java command line. Java classes

```
javac HelloWord.java    // generate HelloWord.class
java HelloWord
```

Jar file, run (nonexecutable JAR) in command line.

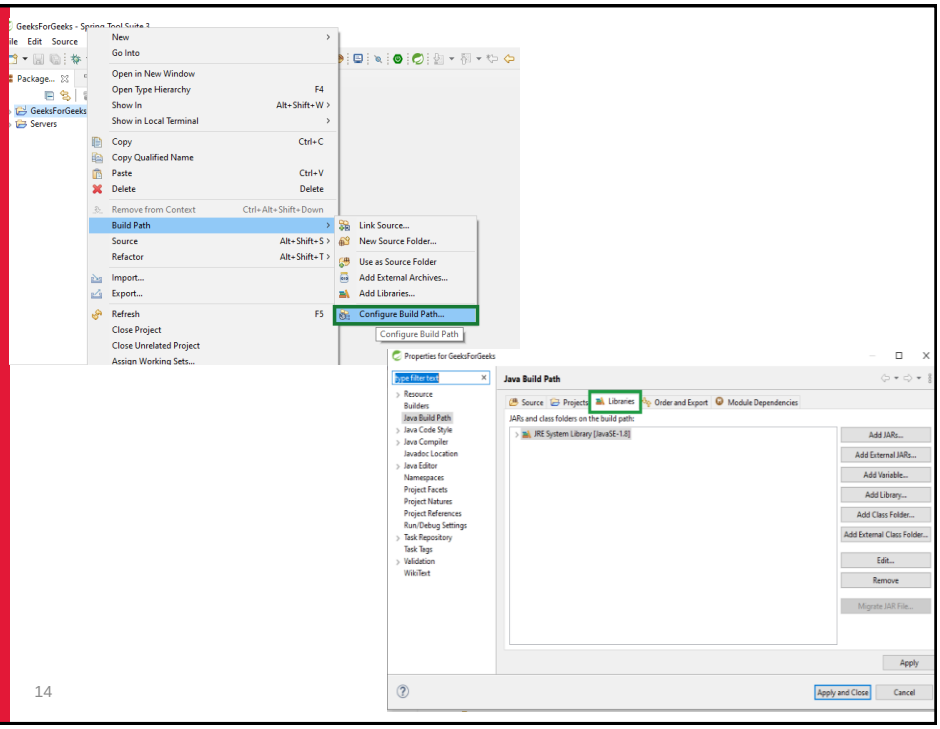
- JAR is an abbreviation of JAVA Archive. It is used for aggregating multiple files into a single one, and it is present in a ZIP format.
  - Put in CLASSPATH
  - In Command line

```
javac -cp  path/abc.jar HelloWord.java
or
javac  -cp  .: path/abc.jar HelloWord.java    ; in windows

java -cp .: path/abc.jar HelloWord.java
```

  - In eclipse
    Add to build path

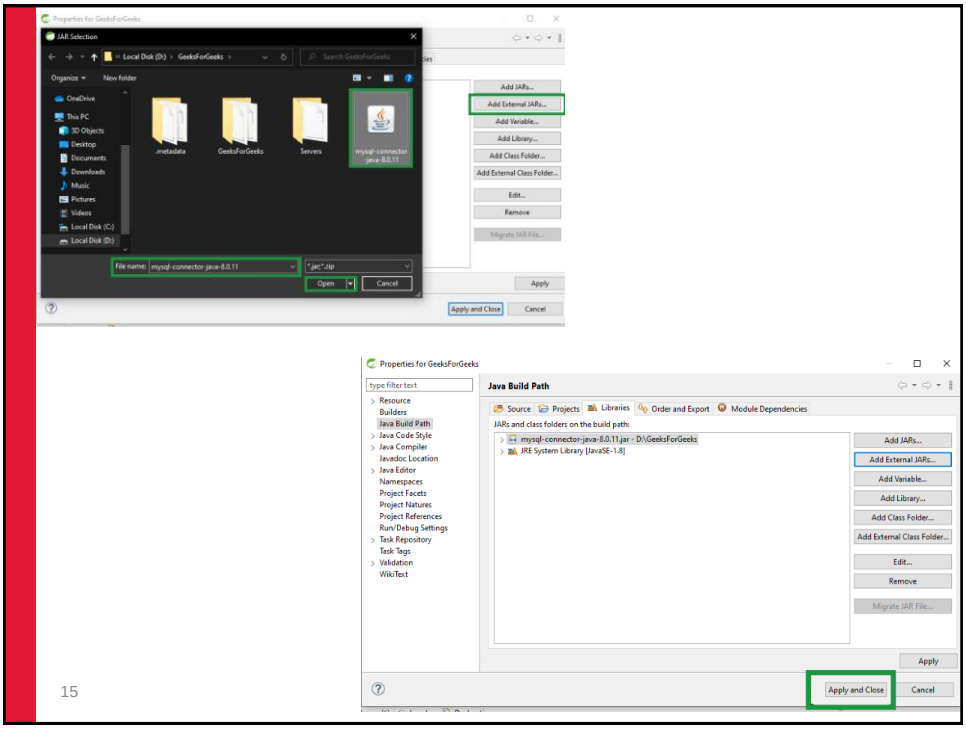
13

13




14

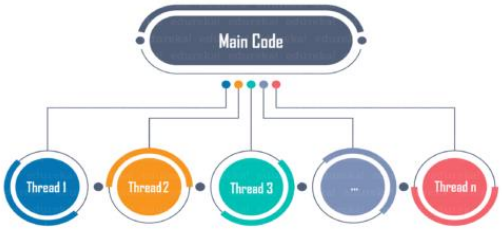
14




15

15

# Multithreading




16

# Multithreading

What is a process? -- a program in execution
What is a thread? -- A thread is an execution stream within a process.

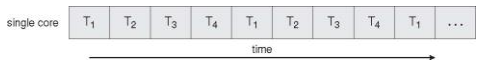
- A thread is also called a "lightweight process".
- Has its own execution stack, local variables, and program counter.
- Very much like a process, but it runs within a process.

Multithreading

- There may be more than one thread in a process.
  - Is called a *multithreaded* process.
  - Multithreading provides the capability to run tasks in parallel for a process.
  - All threads of a process share with each other resources allocated to the
  - process. -- In fact, they compete with each other.

- Threads allow the programmer to turn a program into separate, independently running subtasks
  - E.g., allows a server to handle multiple clients simultaneously.

- In single core CPU, each thread give short time slice. In multiple core CPU, may run in parallel

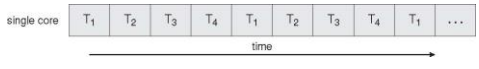



17

17

# What are Threads?

- A piece of code that run in concurrent with other threads.
- Each thread is a statically ordered sequence of instructions.
- Threads are being extensively used express concurrency on both single and multiprocessors machines.




- Programming a task having multiple threads of control – Multithreading or Multithreaded Programming.

18

18

# A single threaded program

```
class ABC
{
  ....
    public void main(..)
    {
    ...
    ..
    }
}
```

begin

body

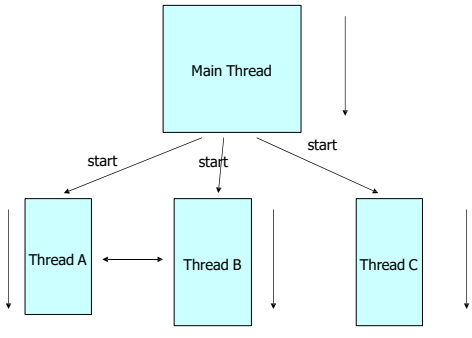
end

"Main thread"





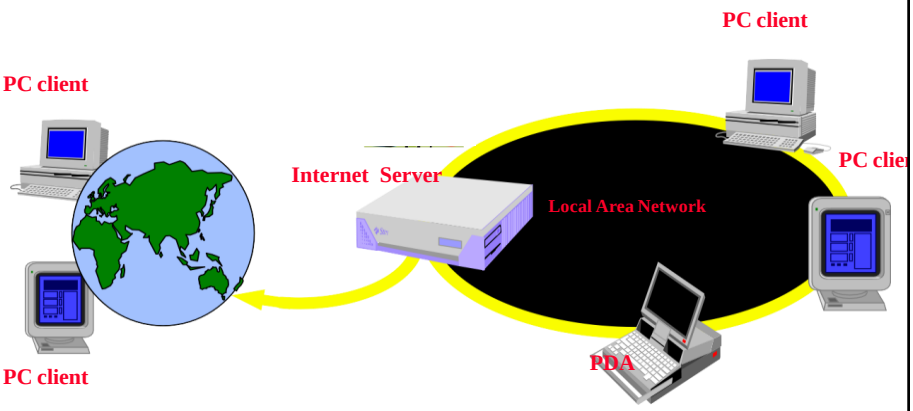
# A Multithreaded program

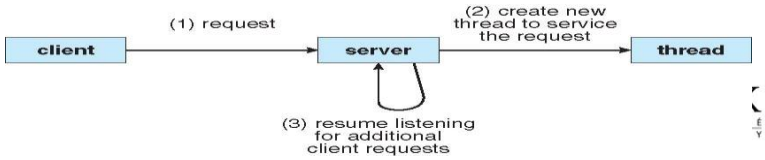



Threads may switch or exchange data/results

Main thread: from which other "child" threads will be spawned.
Often, it must be the last thread to finish execution because it performs various shutdown actions.

# Web/Internet Applications: Serving Many Users Simultaneously




21

21

Java has built in thread support for Multithreading

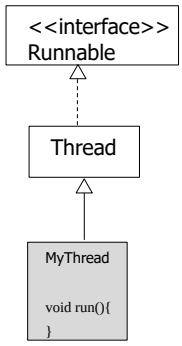
- Thread creation
- Thread Synchronization
- Thread Scheduling
- Inter-Thread Communication

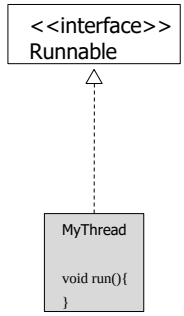
22

22

# Creating thread in Java

- Create a class that extends the **Thread** class
- Create a class that implements the **Runnable** interface




```
Public class MyThread extends Thread
```

```
Public class MyThread implements Runnable
```

23

23

compact1, compact2, compact3
java.lang

## Interface Runnable

**All Known Subinterfaces:**
RunnableFuture<V>, RunnableScheduledFuture<V>

**All Known Implementing Classes:**
AsyncBoxView.ChildState, ForkJoinWorkerThread, FutureTask, RenderableImageProducer, SwingWorker, Thread, TimerTask

**Functional Interface:**
This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

---

```
@FunctionalInterface
public interface Runnable
```

The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. The class must define a method of no arguments called run.

---

compact1, compact2, compact3
java.lang

## Class Thread

java.lang.Object
    java.lang.Thread

**All Implemented Interfaces:**
Runnable

**Direct Known Subclasses:**
ForkJoinWorkerThread

---

```
public class Thread
extends Object
implements Runnable
```

A *thread* is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

24

**The `Thread` class**

```
public class Thread implements Runnable {!
    public Thread();
    public Thread(Runnable target);
    public Thread(String name);
    public Thread(Runnable target, String name);

    public static final int MIN_PRIORITY = 1;
    public static final int NORM_PRIORITY = 5;
    public static final int MAX_PRIORITY = 10;

    public final String getName();
    public final int getPriority();
    public void interrupt();!
    public final native boolean isAlive();
    public void start();   causes this thread to start execution
    public final synchronized void join() throws InterruptedException;
    public void run(); If this thread was constructed using a separate
   Runnable run object, then that Runnable object's run method is calle

    public final void setName(String name);
    public final void setPriority(int newPriority);!
    public String toString();
```

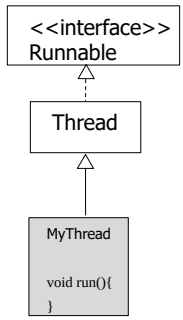
25

25

# 1st method: Extending Thread class

- Create a class by extending Thread class and override **run**() method:

```
class MyThread extends Thread
{
   public void run()
   {
     // thread body of execution
   }
}
```




- Create a thread:

```
MyThread thr1 = new MyThread();
```

- Start Execution of threads:

```
thr1.start();
```

or

- Create and Execute:

```
new MyThread().start();
```

```
class MyThread extends Thread {
      public void run() {
            System.out.println(" this thread is running ... ");
      }
}

class ThreadTest {
      public static void main(String [] args ) {
         MyThread t = new MyThread();
         t.start();
      }
}
```

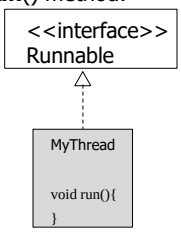
26

26

# 2nd method: Extending Thread class

- Create a class that implements the interface Runnable and override **run**() method:

```
class MyThread implements Runnable
{
   public void run()
   {
     // thread body of execution
   }
}
```




- Creating Object:
  `MyThread myObject = new MyThread();`
- Creating Thread Object:
  `Thread thr1 = new Thread( myObject );`
- Start Execution:
  `thr1.start();`

or

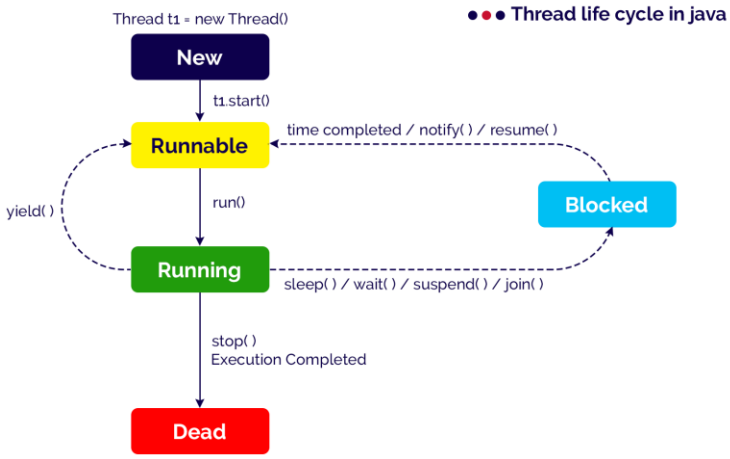
- Create and Execute:
  `new Thread(new MyThread()).start();`

```
class MyThread implements Runnable
      {
      public void run() {
            System.out.println(" is running ... ");
      }
}

class ThreadEx2 {
      public static void main(String [] args ) {
            Thread t = new Thread(new
                                    MyThread());
             t.start();
      }
}
```

27

27

# Life cycle of Thread




www.btechsmartclass.com

28

28

# Multiple threads (same or different)

```
class A extends Thread
{
    public void run()
     {
        for(int i=1; i<=5; i++)
         {
            System.out.println("\t From ThreadA: i= "+i);
         }
         System.out.println("Exit from A");
     }
}
```

```
class B extends Thread
{
    public void run()
     {
        for(int j=1;j<=5;j++)
         {
            System.out.println("\t From ThreadB: j= "+j);
         }
         System.out.println("Exit from B");
     }
}
```

```
class C extends Thread
{
    public void run()
     {
        for(int k=1;k<=5;k++)
         {
           System.out.println("\t From ThreadC: k= "+k);
         }
           System.out.println("Exit from C");
```

```
class ThreadTest
{
    public static void main(String args[])
     {
        new A().start();
        new B().start();
        new C().start();
        System.out.println("main thread ends")
     }
}
```

31

31

## Run 1

```
% java ThreadTest
Main thread ends
        From ThreadA: i= 1
        From  ThreadA:  i=  2
        From  ThreadA:  i=  3
        From  ThreadA:  i=  4
        From ThreadA: i= 5
Exit from A
        From  ThreadC:  k=  1
        From  ThreadC:  k=  2
        From  ThreadC:  k=  3
        From  ThreadC:  k=  4
        From ThreadC: k= 5
Exit from C
        From  ThreadB:  j=  1
        From  ThreadB:  j=  2
        From  ThreadB:  j=  3
        From  ThreadB:  j=  4
        From ThreadB: j= 5
Exit from B
```

## Run 2

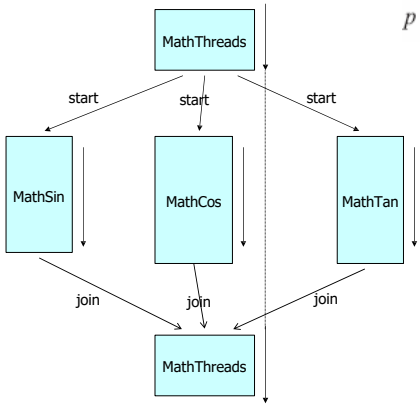
```
% java ThreadTest
Main thread ends
        From ThreadA: i= 1
        From  ThreadA:  i=  2
        From  ThreadA:  i=  3
        From  ThreadA:  i=  4
        From  ThreadA:  i=  5
        From  ThreadC:  k=  1
        From  ThreadC:  k=  2
        From  ThreadC:  k=  3
        From  ThreadC:  k=  4
        From ThreadC: k= 5
Exit from C
        From  ThreadB:  j=  1
        From  ThreadB:  j=  2
        From  ThreadB:  j=  3
        From  ThreadB:  j=  4
        From ThreadB: j= 5
Exit  from  B
Exit from A
```

32

32

# Wait for threads to finish?

$$p = \sin(x) + \cos(y) + \tan(z)$$




do{

}while {(t1.alive() || t2.isAlive() || t3.isAlive() }

t1.join()
t2.join()
t3.join()





```
/* MathThreads.java: A program with multiple threads performing concurrent
operations. */
import java.lang.Math;
class MathSin extends Thread {
   public double deg;
   public double res;

   public MathSin(int degree) {
     deg = degree;
   }
   public void run() {
           System.out.println("Executing sin of "+deg);
      double Deg2Rad = Math.toRadians(deg);
      res = Math.sin(Deg2Rad);
      System.out.println("Exit from MathSin. Res = "+res);
   }
}
class MathCos extends Thread {
   public double deg;
   public double res;

   public MathCos(int degree) {
      deg = degree;
   }
   public void run() {
      System.out.println("Executing cos of "+deg);
      double Deg2Rad = Math.toRadians(deg);
      res = Math.cos(Deg2Rad);
      System.out.println("Exit from MathCos. Res = "+res);
   }
}
class MathTan extends Thread {
   public double deg;
   public double res;

   public MathTan(int degree) {
      deg = degree;
   }
   public void run() {
      System.out.println("Executing tan of "+deg);
      double Deg2Rad = Math.toRadians(deg);
      res = Math.tan(Deg2Rad);
      System.out.println("Exit from MathTan. Res = "+res);
```

34

$$p = \sin(x) + \cos(y) + \tan(z)$$

```
class MathThreads {
   public static void main(String args[]) {
      MathSin st = new MathSin(45);
      MathCos ct = new MathCos(60);
      MathTan tt = new MathTan(30);
      st.start();
      ct.start();
      tt.start();
      try {   // wait for completion of all thread and then sum
        st.join();
        ct.join(); //wait for completion of MathCos object
        tt.join();
        double z = st.res + ct.res + tt.res;
        System.out.println("Sum of sin, cos, tan = "+z);
      }
      catch(InterruptedException IntExp) {
      }
   }
}
```

35

35

```
Run 1:
[raj@mundroo] threads [1:111] java MathThreads
Executing sin of 45.0
Executing cos of 60.0
Executing tan of 30.0

Exit from MathSin. Res = 0.7071067811865475
Exit from MathCos. Res = 0.5000000000000001
Exit from MathTan. Res = 0.5773502691896257
Sum of sin, cos, tan = 1.7844570503761732

Run 2:
[raj@mundroo] threads [1:111] java MathThreads
Executing sin of 45.0
Executing tan of 30.0
Executing cos of 60.0
Exit from MathCos. Res = 0.5000000000000001
Exit from MathTan. Res = 0.5773502691896257
Exit from MathSin. Res = 0.7071067811865475
Sum of sin, cos, tan = 1.7844570503761732

Run 3:
[raj@mundroo] threads [1:111] java MathThreads
Executing cos of 60.0
Executing sin of 45.0
Executing tan of 30.0
Exit from MathCos. Res = 0.5000000000000001
Exit from MathTan. Res = 0.5773502691896257
Exit from MathSin. Res = 0.7071067811865475
Sum of sin, cos, tan = 1.7844570503761732
```

36

36

## Accessing Shared Resources

- Applications Access to Shared Resources need to be coordinated.
  - Printer (two person jobs cannot be printed at the same time)
  - Simultaneous operations on your bank account.
  - Can the following operations be done at the same time on the same account?
    - *Deposit()*
    - *Withdraw()*
    - *Enquire()*

37

37

## Accessing Shared Resources

- Applications Access to Shared Resources need to be coordinated.
  - If one thread tries to read the data and other thread tries to read the same data, there is no issue.
  - If one thread tries to read the data and other thread tries to update the same data, it leads to inconsistent state.
  - This can be prevented by synchronizing access to the data.
  - Use **synchronized** method:

```
public synchronized void update()
{
        …
}
```

38

38

# Shared account object between 3 threads

```
class MyThread implements Runnable  {
    Account account;
    public MyThread (Account s) {  this.account = s;}
    public void run() { account.deposit(); }
} // end class MyThread
```

```
class YourThread implements Runnable {
    Account account;
    public YourThread (Account s) { this.account = s;}
    public void run() { account.withdraw(); }
} // end class YourThread
```

```
class HerThread implements Runnable {
    Account account;
    public HerThread (Account s) { account = s; }
    public void run() {account.enquire(); }
} // end class HerThread
```

35

39

```
class InternetBankingSystem {
    public static void main(String [] args ) {
        Account accountObject = new Account ();
        Thread t1 = new Thread(new MyThread(accountObject));
        Thread t2 = new Thread(new YourThread(accountObject));
        Thread t3 = new Thread(new HerThread(accountObject));
        t1.start();
        t2.start();
        t3.start();
      // DO some other operation
    } // end main()
}
```

account (shared object)

40

34

40

## Monitor (shared object access): serializes operation on shared object

```
class Account {
    int balance;

    // if 'synchronized' is removed, the outcome is unpredictable
    public synchronized void deposit( ) {
        balance += deposit_amount;
    }

    public synchronized void withdraw( ) {
        balance -= deposit_amount;
    }

    public synchronized void enquire( ) {
        System.out.println( this.balance);
    }
}
```

41

41

## More on multi-threading

- Set priority
  - `mythread.setPriority(Thread.MAX_PRIORITY);`
- Use executor thread pool
  - An executor is an object that executes `Runnable` tasks.
  - Separates task submission from execution policy:
  - Use **`anExecutor.execute(aRunnable)!`**
  - Instead of **`new Thread(aRunnable).start();`**

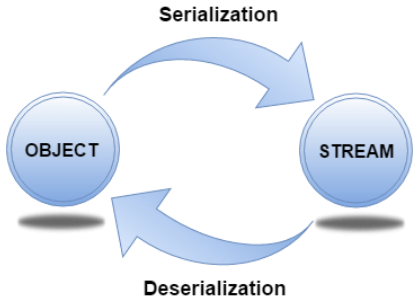
```
ExecutorService executor = Executors.newFixedThreadPool(3);
executor.execute(new myThread());
executor.execute(new myThread2();
executor.execute(new myThread3());
executor.shutdown();
while (!executor.isTerminated())
  ;
```

- Inter-process communication
  - wait() notify() notifyAll()….
- Race condition, deadlock…

42

42

# Object Serilization

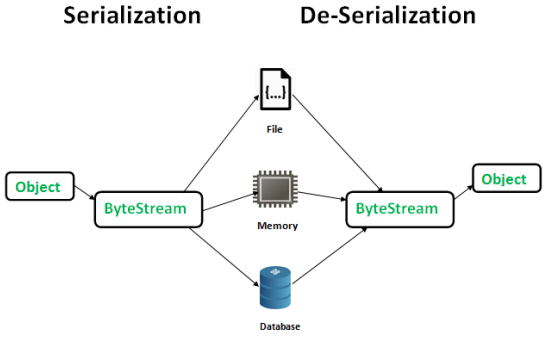



43

43

## Data persistent

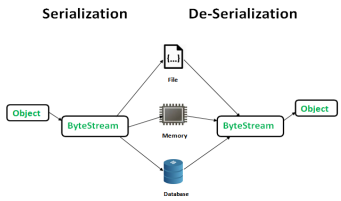
- Save object data ? Save to disk or database
- How: save each field to a (text) file, and later read in field by field and build the object
  - Using *filewitter, bufferwritter, filereader, buffereader, scanner …*
- Easier way: write and read whole object – serialize and deserialize it




44

44

# Serialization




- **Serialization** is a mechanism of converting the state of an object into a byte stream.
- **Deserialization** is the reverse process where the byte stream is used to recreate the actual Java object in memory.
- This mechanism is used to persist the object.

- The byte stream created is platform independent. So, the object serialized on one platform can be deserialized on a different platform.

- To make a Java object serializable we implement the **java.io.Serializable** interface.
- The ObjectOutputStream class contains **writeObject()** method for serializing an Object.

45

45

```java
import java.io.Serializable;

public class Student implements Serializable
{
 int id;
 String name;

 public Student(int id, String name) {
  this.id = id;
  this.name = name;
 }

}
```

```java
import java.io.*;

class Persist{
 public static void main(String args[])throws Exception{
  Student s1 =new Student(211, "ravi");

  FileOutputStream fout=new FileOutputStream("fileOut");
  ObjectOutputStream out=new ObjectOutputStream(fout);

  out.writeObject(s1);
  out.flush();

  System.out.println("success");
 }
}

}
```

46

46

- The ObjectInputStream class contains **readObject()** method for deserializing an object.

```
import java.io.*;

class Depersist{

 public static void main(String args[]){
  try{
  //Creating stream to read the object
  ObjectInputStream in=new ObjectInputStream(new FileInputStream("fileOut"));

  Student s= (Student) in.readObject();

  //printing the data of the serialized object
  System.out.println(s.id+" "+s.name);

  //closing the stream
  in.close();

  }catch(Exception e){System.out.println(e);}

 }

}
```

211 ravi

47

47

# If don't want to serialize a field -- transient

```
import java.io.Serializable;

public class Student implements Serializable
{
 int id;
 transient String name;

 public Student(int id, String name) {
  this.id = id;
  this.name = name;
 }

}
```

```
import java.io.*;

class Persist{

 public static void main(String args[])throws Exception{

  Student s1 =new Student(211,"ravi");

  FileOutputStream fout=new FileOutputStream("fileOut");
  ObjectOutputStream out=new ObjectOutputStream(fout);

  out.writeObject(s1);

  out.flush();

  System.out.println("success");
 }
}
```

A variable defined with transient keyword is not serialized during serialization process.

This variable will be initialized with default value during deserialization. (e.g: for objects it is null, for int it is 0).

48

- The ObjectInputStream class contains **readObject()** method for deserializing an object.

```
import java.io.*;

class Depersist{

 public static void main(String args[]){
  try{
  //Creating stream to read the object
  ObjectInputStream in=new ObjectInputStream(new FileInputStream("fileOut"));

  Student s= (Student) in.readObject();

  //printing the data of the serialized object
  System.out.println(s.id+" "+s.name);

  //closing the stream
  in.close();

  }catch(Exception e){System.out.println(e);}

 }

}
```

211 null

49

49

# Serializable classes in Java

java.lang

**Class String**

java.lang.Object
    java.lang.String

**All Implemented Interfaces:**

Serializable, CharSequence, Comparable<String>

```
public final class String
extends Object
implements Serializable, Comparable<String>, CharSequence
```

If an attribute is a object, that object should be serializable too, otherwise run time exception.

String is such an example

java.lang

**Class Number**

java.lang.Object
    java.lang.Number

**All Implemented Interfaces:**

Serializable

**Direct Known Subclasses:**

AtomicInteger, AtomicLong, BigDeci
LongAdder, Short

```
public abstract class Number
extends Object
implements Serializable
```

java.lang

**Class Integer**

java.lang.Object
    java.lang.Number
        java.lang.Integer

**All Implemented Interfaces:**

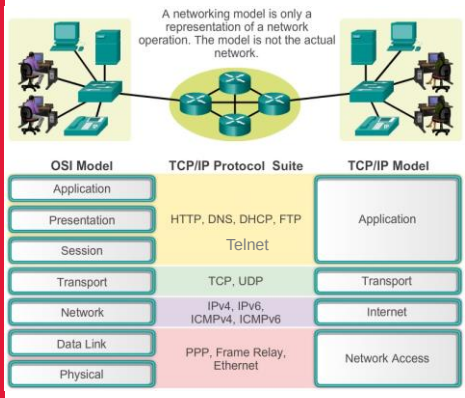
Serializable, Comparable<Integer>

```
public final class Integer
extends Number
implements Comparable<Integer>
```

50

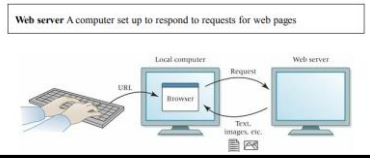
50

# Under the hood – network layers




UDP: Unreliable: messages may be lost or reordered

TCP: other end is notified if packet lost, or one end closes or resets the connection, o




HTTP over TCP/IP

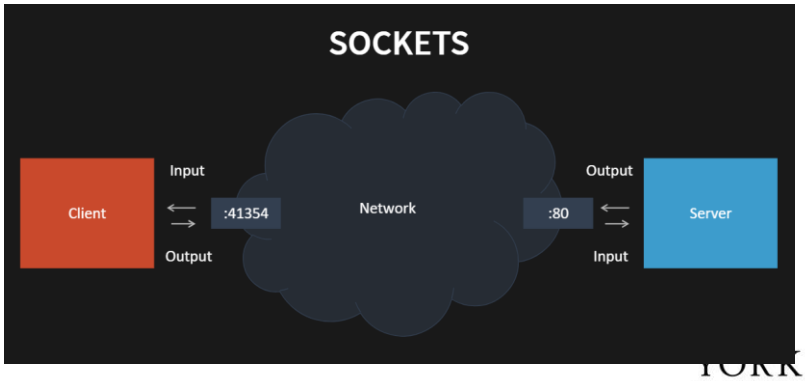
1. Reformat the URL entered as a valid HTTP request message.
   - If the server is specified using a host name (rather than an IP address), use DNS to convert this name to the appropriate IP address.
2. **Establish a TCP connection using the IP address of the specified web server.**
3. Send the HTTP request over the TCP connection and wait for the server's response.
4. Display the document contained in the response. If the document is not a plain-text document but instead is written in a language such as HTML, this involves *rendering* the document

51

51

# TCP connection: Socket

Goal: learn how to build client/server application that communicate using sockets




58

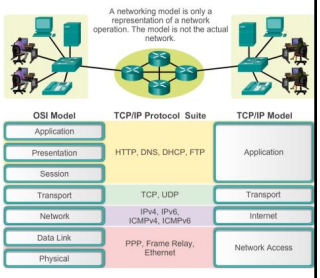
58

# Socket programming

**Socket**: a *host-local*, *application-created*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process

**Socket API**

- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram (UDP)
  - reliable, byte stream-oriented (TCP)

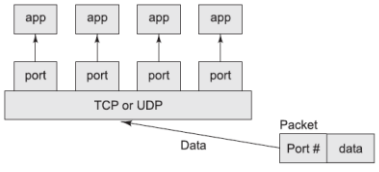







# Some jargons

- IP Address
  - Names a machine/host.
- Port
  - Port maps to one "channel" on a host.




- A stream is a sequence of characters that flow into or out of a process.
- An input stream is attached to some input source for the process, e.g., keyboard or socket.
- An output stream is attached to an output source, e.g., monitor or socket

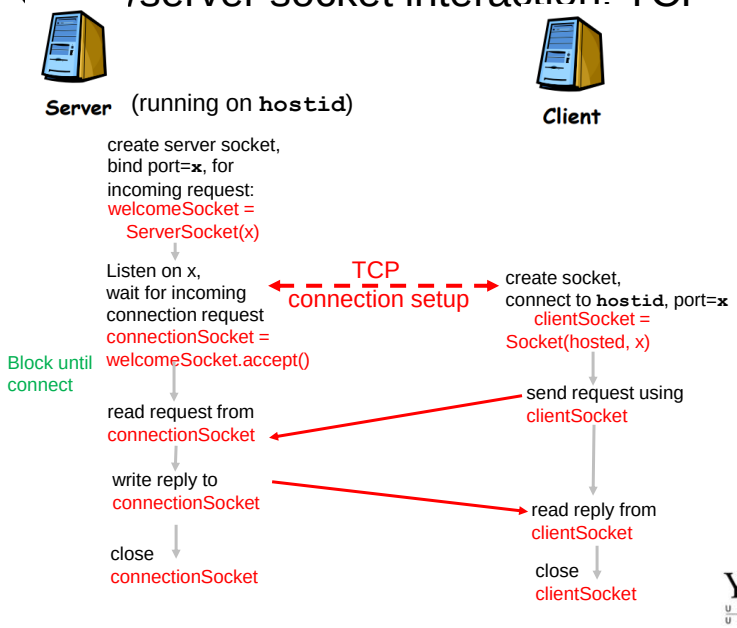
# `java.net` Package

- A package that provides classes and an API for socket programming
  - Open/close socket.
  - Accept – listens for a connection
  - Read/Write.
  - Send/Receive.

- See the classes:
  - **ServerSocket** – used by server to listen for clients.
  - **Socket** – used by both server and client. Represents a connection.

61

# Client/server socket interaction: TCP




62

# Example: TCP Java server (v0)

Server

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception    {
     String clientSentence;
     String replySentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     Socket connectionSocket = welcomeSocket.accept();

     BufferedReader inFromClient = new BufferedReader(new
          InputStreamReader(connectionSocket.getInputStream()));

     PrintWriter outToClient =
          new PrintWriter(connectionSocket.getOutputStream());

     clientSentence = inFromClient.readLine();

     outToClient.println("Hi: " + clientSentence);

     outToClient.close();  // flush()
}}
```

Create welcoming socket Bind at port 6789

Listen, wait, on welcoming socket for contact by client

Create input stream, attached to socket

Create output stream, attached to socket

Read in line from socket

Write out line to socket

YORK UNIVERSITÉ UNIVERSITY

63

63

# Example: TCP Java server (v0.1)

Server

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception    {
     String clientSentence;
     String replySentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     Socket connectionSocket = welcomeSocket.accept();

     BufferedReader inFromClient = new BufferedReader(new
          InputStreamReader(connectionSocket.getInputStream()));

     PrintWriter outToClient =
          new PrintWriter(connectionSocket.getOutputStream());

     clientSentence = inFromClient.readLine();

     replySentence =  clientSentence.toUpperCase();
     outToClient.println("Hi: " + replySentence);

     outToClient.close(); // flush()
}}
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

Create output stream, attached to socket

Read in line from socket

Write out line to socket

2 stream

64

64

# Connect to the server

- TELNET ***$ telnet hostname port***
  - Telnet is a client-server protocol based on text-oriented data exchange over TCP connections.
  - Telnet enables remote communication with a TCP server via text-based inputs and outputs.

```
yu266074@yu266074-HP-Pavilion-TS-Sleekbook-14:~$ telnet 192.168.0.13 6789
Trying 192.168.0.13...
Connected to 192.168.0.13.
Escape character is '^]'.
how are you
Hi: HOW ARE YOU
Connection closed by foreign host.
yu266074@yu266074-HP-Pavilion-TS-Sleekbook-14:~$
```

- Browser/curl: send HTTP header to process (next)
  - Need to handle header
- A java socket client program (next)

65

65

# Example: Java client (TCP)

Client

```
import java.io.*;
import java.net.*;
class TCPClient {
    public static void main(String argv[]) throws Exception
    {
        String sentence;
        String modifiedSentence;

        BufferedReader inFromUser =
          new BufferedReader(new InputStreamReader(System.in));

        Socket clientSocket = new Socket("somehostname", 6789);

        PrintWriter outToServer =
          new PrintWriter(clientSocket.getOutputStream(), true);

        BufferedReader inFromServer =
          new BufferedReader(new
          InputStreamReader(clientSocket.getInputStream()));

        sentence = inFromUser.readLine();
        outToServer.println(sentence');

        modifiedSentence = inFromServer.readLine();

        System.out.println("FROM SERVER: " + modifiedSentence);

        clientSocket.close();
```

Create input stream

Create client socket, connect to server

Create output stream attached to socket

Create input stream attached to socket

Send line to server

Read line from server

3 stream

66

# Example: TCP Java server (v0.5)

Server

Continue commination

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception    {
     String clientSentence;
     String replySentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     Socket connectionSocket = welcomeSocket.accept();

     BufferedReader inFromClient = new BufferedReader(new
          InputStreamReader(connectionSocket.getInputStream()));

     PrintWriter outToClient =
          new PrintWriter(connectionSocket.getOutputStream());

     while (clientSentence = inFromClient.readLine() != "done"){

        replySentence =  clientSentence.toUpperCase();
        outToClient.println("Hi: " + replySentence);
     }
  }
```

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

Create output stream, attached to socket

Read in line from socket

Write out line to socket





# Example: Java client (v0.5)

Client

```
import java.io.*;
import java.net.*;
class TCPClient {
   public static void main(String argv[]) throws Exception
   {
      String sentence;
      String modifiedSentence;

      BufferedReader inFromUser =
        new BufferedReader(new InputStreamReader(System.in));

      Socket clientSocket = new Socket ("somehostname", 6789);

      PrintWriter outToServer =
        new PrintWriter(clientSocket.getOutputStream(), true);

      BufferedReader inFromServer = new BufferedReader(new
       InputStreamReader(clientSocket.getInputStream()));

      while(sentence = inFromUser.readLine() != null);
        outToServer.println(sentence);
        if  (sentence.equals("done"))  break;
        modifiedSentence = inFromServer.readLine();

       System.out.println("FROM SERVER: " + modifiedSentence);
      }
      clientSocket.close();
```

Create input stream

Create client socket, connect to server

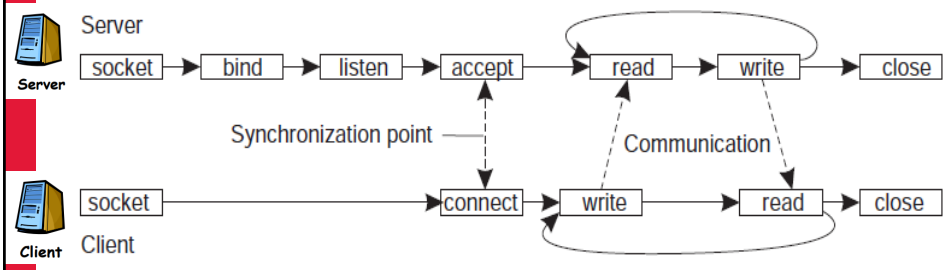
Create output stream attached to socket

Create input stream attached to socket

Send line to server

Read line from server

# TCP Sockets




Socket server: can do calculations/services requested by client

- can retrieve data from database if needed
- can create objects and send over
  - **ObjectOutputStream outObj=new ObjectOutputStream (clientSocket.getOutputStream())**
  - **outObj.write(objStu);**

  - **ObjectInputStream inObj=new ObjectInputStream(echoSocket.getInputStream()) ;**
  - **Student s= (Student) inObj.readObject();**

Improvement: ?

70

```
System.out.print("first number: ");
userInput = stdIn.readLine();
out.println(userInput);
System.out.println("echo: " + in.readLine());

System.out.print("second number: ");
userInput = stdIn.readLine();
out.println(userInput);
System.out.println("echo: " + in.readLine());


{   Thread.sleep(0);
    Result s= (Result) inObj.readObject();
    System.out.println("received object. sum: "+s.getSum()+"  diff: "+s.getDiff());
}
```

```
inputLine = in.readLine();
System.out.println("received " + inputLine);
out.println(inputLine);
int a = Integer.parseInt(inputLine);

inputLine = in.readLine();
System.out.println("received " + inputLine);
out.println(inputLine);
int b = Integer.parseInt(inputLine);


Result s = new Result( a+b, a-b );
outObj.writeObject(s);
```

```
class Result implements Serializable  {

    private int sum;
    private int diff;

    public Result(int s, int d) {
        this.sum = s;
        this.diff = d;
```

71

71

# Example: TCP Java server (v1)

Create welcoming socket at port 6789

Wait, on welcoming socket for contact by client

Create input stream, attached to socket

Create output stream, attached to socket

Read in line from socket

Write out line to socket

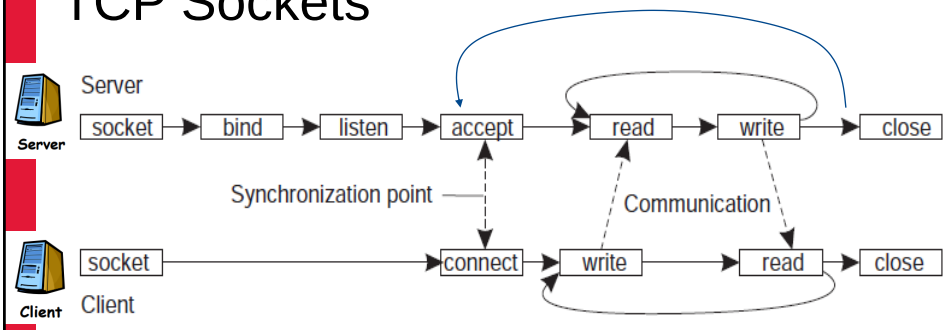
```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception    {
     String clientSentence;
     String replySentence;

     ServerSocket welcomeSocket = new ServerSocket(6789);

     while(true) {   // keep on running

       Socket connectionSocket = welcomeSocket.accept();

        BufferedReader inFromClient = new BufferedReader(new
              InputStreamReader(connectionSocket.getInputStream()));

        PrintWriter outToClient =
              new PrintWritter(connectionSocket.getOutputStream());

        while (!(clientSentence = inFromClient.readLine()).equals("done")){

          replySentence =  clientSentence.toUpperCase();
          outToClient.println("Hi" + replySentence);
          outToClient.flush();  // or set true
       }
     }
```

YORK UNIVERSITÉ UNIVERSITY

72

72

# TCP Sockets




More Improvement: ?

YORK UNIVERSITÉ UNIVERSITY

73

# Example: TCP Java server (v2)

```
import java.io.*;
import java.net.*;

class TCPServer {

  public static void main(String argv[]) throws Exception    {

    ServerSocket welcomeSocket = new ServerSocket(6789);

    while(true) {   // keep on running

      Socket connectionSocket = welcomeSocket.accept();

      ClientHandler handler = new ClientHandler
                                        (connectionSocket);
      handler.start();
  }}
```

```
import java.io.*;
import java.net.*;

class ClientHandler extends Thread {

   Socket clientSocket;
   String clientSentence;
   String replySentence;

   public ClientHandler (Socket s){
      this.clientSocket = s;
   }


   public void run(){

     BufferedReader inFromClient = new BufferedReader(new
            InputStreamReader(clientSocket.getInputStream()));

      PrintWriter outToClient =
             new PrintWritter(clientSocket.getOutputStream());

      while (!(clientSentence = inFromClient.readLine()).equals("done“)){

         replySentence =  clientSentence.toUpperCase();
         outToClient.println("Hi" + replySentence);
         outToClient.flush();  // or set true
      }   // close socket, streams…
```

74

74

# Case study  more on Socket

| | Method | URL or Pathname | HTTP version | Headers | Message body |
|---|---|---|---|---|---|
| Request | GET | https://www.linkedin.com/jobs/ | HTTP/ 1.1 | | |

| | HTTP version | Status code | Reason | Headers | Message body |
|---|---|---|---|---|---|
| Response | HTTP/ 1.1 | 200 | OK | | Resource data |

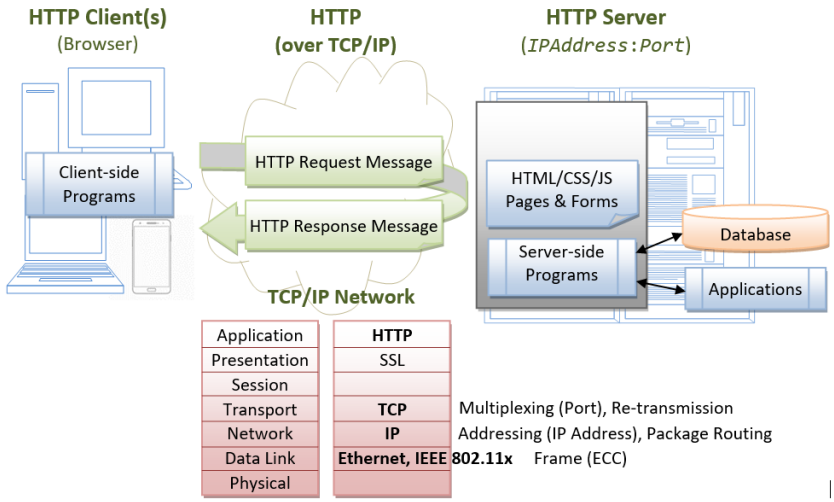
- Generally, any message and process
- As a HTTP client
  - send http request to website. Simulates browser
- As a HTTP server
  - handle http request, simulate as a website
  - client: browser or curl
  - return:  200 status, file content
  - if not, error message 403 404 500 etc

**HTTP STATUS CODES**

| | |
|---|---|
| **2xx Success** | |
| 200 | Success / OK |
| **3xx Redirection** | |
| 301 | Permanent Redirect |
| 302 | Temporary Redirect |
| 304 | Not Modified |
| **4xx Client Error** | |
| 401 | Unauthorized Error |
| 403 | Forbidden |
| 404 | Not Found |
| 405 | Method Not Allowed |
| **5xx Server Error** | |
| 501 | Not Implemented |
| 502 | Bad Gateway |
| 503 | Service Unavailable |
| 504 | Gateway Timeout |

75

75




77

77

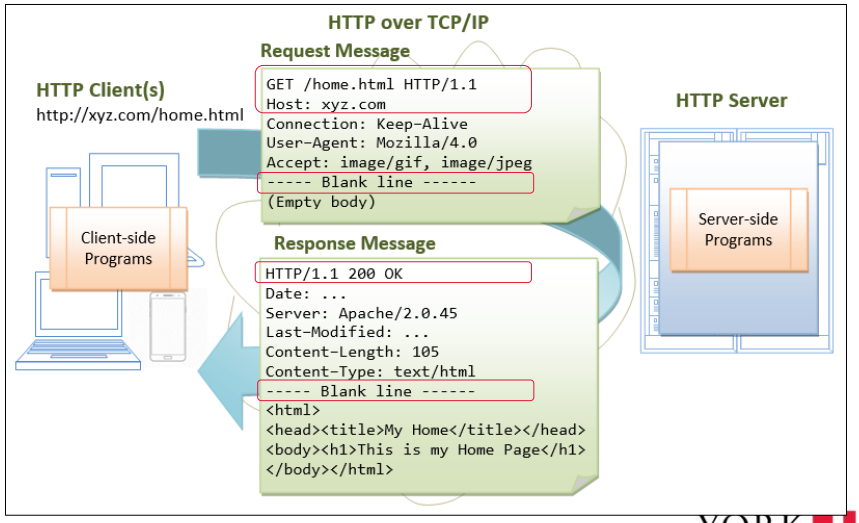
# HTTP

## GET and POST

- GET and POST allow information to be sent back to the web server from a browser
  - e.g. when you click on the “submit” button of a form the data in the form is send back to the server, as "name=value" pairs.
- Choosing GET as the "method" will append all of the data to the URL and it will show up in the URL bar of your browser.
  - The amount of information you can send back using a GET is restricted as URLs can only be 1024 characters.
- A POST sends the information through a socket back to the webserver and it won't show up in the URL bar.
  - This allows a lot more information to be sent to the server
  - The data sent back is not restricted to textual data and it is possible to send files and binary data such as serialized Java objects.

78 of 99

78




79

79

# Case study more on Socket

- As a HTTP client
  - send http request to website. Simulate browser

| | Method | URL or Pathname | HTTP version | Headers | Message body |
|---|---|---|---|---|---|
| Request | GET | https://www.linkedin.com/jobs/ | HTTP/ 1.1 | | |

| | HTTP version | Status code | Reason | Headers | Message body |
|---|---|---|---|---|---|
| Response | HTTP/ 1.1 | 200 | OK | | Resource data |

**Method Call**

**GET** / HTTP 1.1
Host: *server host name*
User-Agent: *software making the request*
Content-Type: text/xml
Content-Length: *number of bytes in payload*

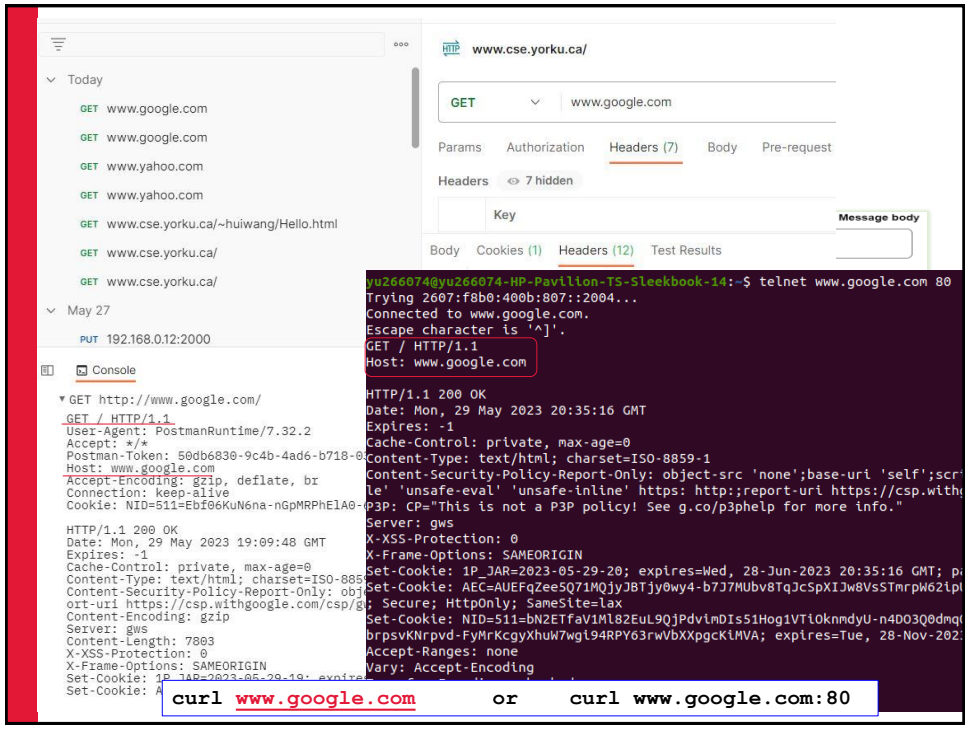
*payload*

**Method Response**

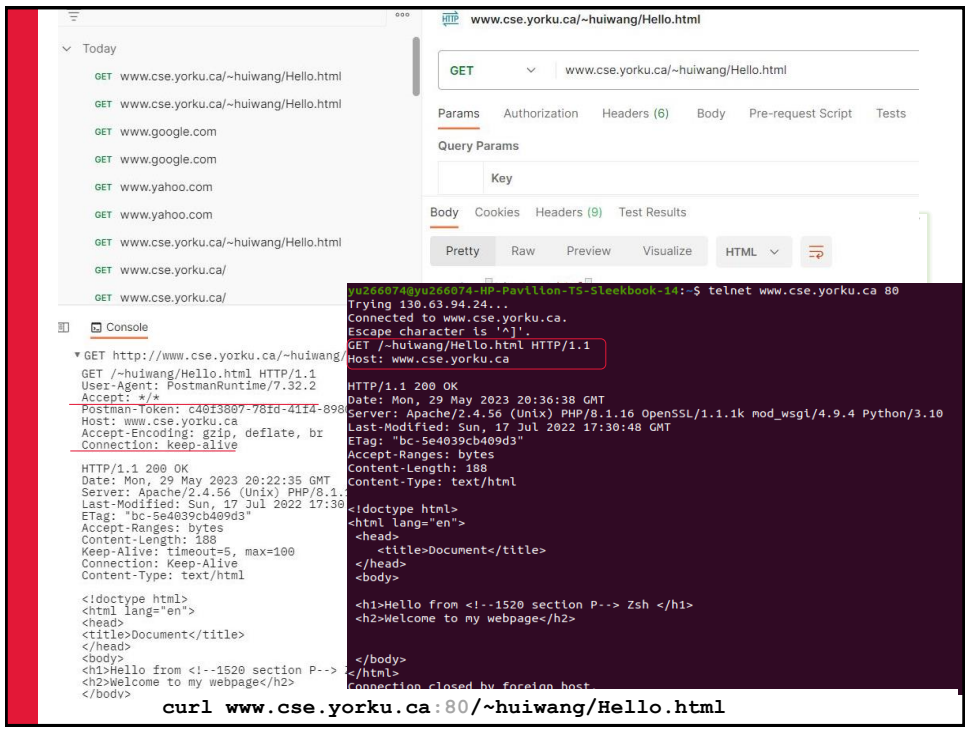
**HTTP1.1 200 OK**
Content-Type: text/xml
Content-Length: *number of bytes in payload*

*payload*

80


Today
GET www.google.com
GET www.google.com
GET www.yahoo.com
GET www.yahoo.com
GET www.cse.yorku.ca/~huiwang/Hello.html
GET www.cse.yorku.ca/
GET www.cse.yorku.ca/
May 27
PUT 192.168.0.12:2000
Console
GET http://www.google.com/
GET / HTTP/1.1
User-Agent: PostmanRuntime/7.32.2
Accept: */*
Host: www.google.com
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
HTTP/1.1 200 OK
Date: Mon, 29 May 2023 19:09:48 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859
Content-Security-Policy-Report-Only: obj
ort-uri https://csp.withgoogle.com/csp/g
Content-Encoding: gzip
Server: gws
Content-Length: 7803
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
www.cse.yorku.ca/
GET
www.google.com
Params
Authorization
Headers (7)
Body
Pre-request
Headers
7 hidden
Key
Message body
Body
Cookies (1)
Headers (12)
Test Results
yu266074@yu266074-HP-Pavilion-TS-Sleekbook-14:~$ telnet www.google.com 80
Trying 2607:f8b0:400b:807::2004...
Connected to www.google.com.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.google.com
HTTP/1.1 200 OK
Date: Mon, 29 May 2023 20:35:16 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-1
Content-Security-Policy-Report-Only: object-src 'none';base-uri 'self';scr
le' 'unsafe-eval' 'unsafe-inline' https: http:;report-uri https://csp.with
P3P: CP="This is not a P3P policy! See g.co/p3phelp for more info."
Server: gws
X-XSS-Protection: 0
X-Frame-Options: SAMEORIGIN
Set-Cookie: 1P_JAR=2023-05-29-20; expires=Wed, 28-Jun-2023 20:35:16 GMT;
Set-Cookie: AEC=AUEFqZeeSQ71MQjyJBTjy0wy4-b7J7MUbv8TqJcSpXIJw8VsSTmrpW62i
; Secure; HttpOnly; SameSite=lax
Set-Cookie: NID=511=bN2ETfaV1Ml82EuL9QjPdvimDIs51Hog1VTiOknmdyU-n4DO3Q0dm
brpsvKNrpvd-FyMrKcgyXhuW7wgi94RPY63rwVbXXpgcKiMVA; expires=Tue, 28-Nov-202
Accept-Ranges: none
Vary: Accept-Encoding
curl www.google.com or curl www.google.com:80

81


Today
GET www.cse.yorku.ca/~huiwang/Hello.html
GET www.cse.yorku.ca/~huiwang/Hello.html
GET www.google.com
GET www.google.com
GET www.yahoo.com
GET www.yahoo.com
GET www.cse.yorku.ca/~huiwang/Hello.html
GET www.cse.yorku.ca/
GET www.cse.yorku.ca/
Console
GET http://www.cse.yorku.ca/~huiwang/
GET /~huiwang/Hello.html HTTP/1.1
User-Agent: PostmanRuntime/7.32.2
Accept: */*
Host: www.cse.yorku.ca
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
HTTP/1.1 200 OK
Date: Mon, 29 May 2023 20:22:35 GMT
Server: Apache/2.4.56 (Unix) PHP/8.1.
Last-Modified: Sun, 17 Jul 2022 17:30
ETag: "bc-5e4039cb409d3"
Accept-Ranges: bytes
Content-Length: 188
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html
<!doctype html>
<html lang="en">
<head>
<title>Document</title>
</head>
<body>
<h1>Hello from <!--1520 section P--> Z
<h2>Welcome to my webpage</h2>
</body>
www.cse.yorku.ca/~huiwang/Hello.html
GET
www.cse.yorku.ca/~huiwang/Hello.html
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Query Params
Key
Body
Cookies
Headers (9)
Test Results
Pretty
Raw
Preview
Visualize
HTML
yu266074@yu266074-HP-Pavilion-TS-Sleekbook-14:~$ telnet www.cse.yorku.ca 80
Trying 130.63.94.24...
Connected to www.cse.yorku.ca.
Escape character is '^]'.
GET /~huiwang/Hello.html HTTP/1.1
Host: www.cse.yorku.ca
HTTP/1.1 200 OK
Date: Mon, 29 May 2023 20:36:38 GMT
Server: Apache/2.4.56 (Unix) PHP/8.1.16 OpenSSL/1.1.1k mod_wsgi/4.9.4 Python/3.10
Last-Modified: Sun, 17 Jul 2022 17:30:48 GMT
ETag: "bc-5e4039cb409d3"
Accept-Ranges: bytes
Content-Length: 188
Content-Type: text/html
<!doctype html>
<html lang="en">
<head>
<title>Document</title>
</head>
<body>
<h1>Hello from <!--1520 section P--> Zsh </h1>
<h2>Welcome to my webpage</h2>
</body>
</html>
curl www.cse.yorku.ca:80/~huiwang/Hello.html

82

# Case study  more on Socket

- As a HTTP client
  - send http request to website. Simulate browser

1. Create a socket connection at 8080
2. Send two lines to the host

   ***GET  /  HTTP/1.1***
   ***Host: hostname***

   or

   ***GET  /path/file  HTTP/1.1*** // if not root
   ***Host: hostname***

3. Read response line by line

83

83

# Example: WebGet (HTTP client)

```java
public static void main(String[] args) throws IOException
{
   // Get command-line arguments

   String host;
   String resource;

   if (args.length == 2)
   {
      host = args[0];
      resource = args[1];
   }
   else  {
       System.out.println("need host and resource /..");
   }

  // Open socket
   final int HTTP_PORT = 80;
   try (Socket s = new Socket(host, HTTP_PORT))
   {
      // Get streams
      InputStream instream = s.getInputStream();
      OutputStream outstream = s.getOutputStream();

      // Turn streams into scanners and writers
      Scanner in = new Scanner(instream);
      PrintWriter out = new PrintWriter(outstream);

      // Send command, for debug
      System.out.println("GET " + resource + " HTTP/1.1");
      System.out.println("Host: " + host + "\n");

      //real work, send to server
      out.println("GET " + resource + " HTTP/1.1");
      out.println("Host: " + host);
      out.println();

      out.flush();

      // Read server response
      while (in.hasNextLine())
      {
         String input = in.nextLine();
         System.out.println(input);
      }
```

```
indigo 372 % java WebGet www.google.ca /
GET / HTTP/1.1
Host: www.google.ca

HTTP/1.1 200 OK
Date: Mon, 29 May 2023 22:09:13 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=ISO-8859-
Content-Security-Policy-Report-Only: objec
9UN9_eXJt6d6WDZg' 'strict-dynamic' 'report
```

```
indigo 374 % java WebGet www.cse.yorku.ca /~huiwang/Hello.html
GET /~huiwang/Hello.html HTTP/1.1
Host: www.cse.yorku.ca

HTTP/1.1 200 OK
Date: Mon, 29 May 2023 22:10:30 GMT
Server: Apache/2.4.56 (Unix) PHP/8.1.16 OpenSSL/1.1.1k mod_wsgi/
Last-Modified: Sun, 17 Jul 2022 17:30:48 GMT
ETag: "bc-5e4039cb409d3"
Accept-Ranges: bytes
Content-Length: 188
Content-Type: text/html

<!doctype html>
<html lang="en">
 <head>
    <title>Document</title>
 </head>
 <body>

 <h1>Hello from <!--1520 section P--> Zsh </h1>
 <h2>Welcome to my webpage</h2>


 </body>
</html>
```

84

# Example: WebGet (HTTP client)

```java
public static void main(String[] args) throws IOException
{
   // Get command-line arguments

   String host;
   String resource;

   if (args.length == 2)
   {
      host = args[0];
      resource = args[1];
   }
   else  {
       System.out.println("need host and resource /..");
   }

  // Open socket
   final int HTTP_PORT = 80;
   try (Socket s = new Socket(host, HTTP_PORT))
   {
      // Get streams
      InputStream instream = s.getInputStream();
      OutputStream outstream = s.getOutputStream();

      // Turn streams into scanners and writers
      Scanner in = new Scanner(instream);
      PrintWriter out = new PrintWriter(outstream);

      // Send command, for debug
      System.out.println("GET " + resource + " HTTP/1.1");
      System.out.println("Host: " + host + "\n");

      //real work, send to server
      out.println("GET " + resource + " HTTP/1.1");
      out.println("Host: " + host);
      out.println();

      out.flush();

      // Read server response
      while (in.hasNextLine())
      {
         String input = in.nextLine();
         System.out.println(input);
      }
```

```
red 355 % java WebGet www.cse.yorku.ca /~abc
GET /~abc HTTP/1.1
Host: www.cse.yorku.ca

HTTP/1.1 404 Not Found
Date: Sun, 04 Jun 2023 18:02:27 GMT
Server: Apache/2.4.56 (Unix) PHP/8.1.16 OpenSSL/1.1.1k mod_
0
Content-Length: 329
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 was not found on this server.</p>
<hr>
<address>Apache/2.4.56 (Unix) PHP/8.1.16 OpenSSL/1.1.1k mod_
10 Server at www.cse.yorku.ca Port 80</address>
</body></html>
red 356 %
```

YORK UNIVERSITÉ UNIVERSITY

85

85

# Case study  more on Socket

- As a HTTP server
  - Listen to and handle *GET / HTTP1.1* request header from browser
  - Send status and content

| | Method | URL or Pathname | HTTP version | Headers | Message body |
|---|---|---|---|---|---|
| Request | GET | https://www.linkedin.com/jobs/ | HTTP/ 1.1 | | |

| | HTTP version | Status code | Reason | Headers | Message body |
|---|---|---|---|---|---|
| Response | HTTP/ 1.1 | 200 | OK | | Resource data |

**Method Call**

```
GET  / HTTP 1.1
Host: server host name
User-Agent: software making the request
Content-Type: text/xml
Content-Length: number of bytes in payload

payload
```

**Method Response**

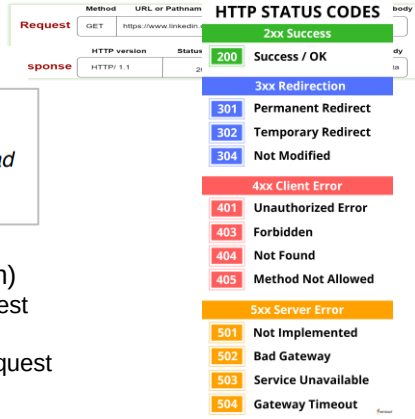
```
HTTP1.1 200 OK
Content-Type: text/xml
Content-Length: number of bytes in payload

payload
```

**HTTP STATUS CODES**

| | |
|---|---|
| **2xx Success** | |
| 200 | Success / OK |
| **3xx Redirection** | |
| 301 | Permanent Redirect |
| 302 | Temporary Redirect |
| 304 | Not Modified |
| **4xx Client Error** | |
| 401 | Unauthorized Error |
| 403 | Forbidden |
| 404 | Not Found |
| 405 | Method Not Allowed |
| **5xx Server Error** | |
| 501 | Not Implemented |
| 502 | Bad Gateway |
| 503 | Service Unavailable |
| 504 | Gateway Timeout |

89

# Case study more on Socket




**Method Response**

> **HTTP1.1 200 OK**
> Content-Type: text/xml
> Content-Length: *number of bytes in payload*
>
> *payload*

- As a HTTP server (lab2, provided program)
  - Listen to and handle *GET / HTTP1.1* request header from browser (or other http client)
  - Listen to and handle *HEAD / HTTP1.1* request header from browser (or other http client)
  - If asking for root, show /index.html
  - If requesting an existing file, give 200 code and return content of the file
  - If file does not exist, give error 404 code and content of '404.html'
  - If ask for other method, such as POST, give 501 code and content of no_support.
  - Support multiple clients
  - There must be a line between header and payload!!!

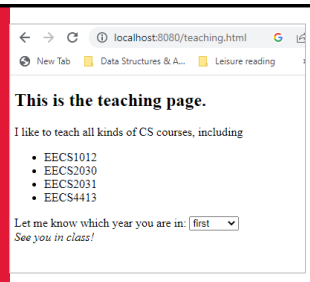
91

```
static final String DEFAULT_FILE = "index.html";
static final String FILE_NOT_FOUND = "404.html";
static final String METHOD_NOT_SUPPORTED = "no_support.html";
try {
    // we read characters from the client via input stream on the socket
    in = new BufferedReader(new InputStreamReader(connect.getInputStream()));

    out = new PrintWriter(connect.getOutputStream());

    // get first line of the request from the client
    String input = in.readLine();
    // we parse the request with a string tokenizer
    StringTokenizer parse = new StringTokenizer(input);
    String method = parse.nextToken().toUpperCase(); // we get the HTTP method of the client
    // we get file requested
    fileRequested = parse.nextToken().toLowerCase();

    // we support only GET and HEAD methods, we check
    if (!method.equals("GET")  &&  !method.equals("HEAD")) {



        // we return the not supported file to the client
        File file = new File(WEB_ROOT, METHOD_NOT_SUPPORTED);
        int fileLength = (int) file.length();
        String contentMimeType = "text/html";


        // we send HTTP Headers with data to client
        out.println("HTTP/1.1 501 Not Implemented");
        out.println("Date: " + new Date());
        out.println("Content-Type: " + contentMimeType);
        out.println("Content-Length: " + fileLength);
        out.println(); // blank line between headers and content, very important !
        out.flush(); // flush character output stream buffer

        // rad file line by line and write to out
        BufferedReader rf = new BufferedReader(new FileReader (file));
        String line;
        while ( (line = ...)
```

92

```java
} else {
    // GET or HEAD method
    if (fileRequested.endsWith("/")) {
        fileRequested += DEFAULT_FILE;
    }

    File file = new File(WEB_ROOT, fileRequested);
    if (! file.exists()){

        file = new File(WEB_ROOT, FILE_NOT_FOUND);
        int fileLength = (int) file.length();
        String contentMimeType = "text/html";

        //byte[] fileData = readFileData(file, fileLength);

        out.println("HTTP/1.1 404 Not Found");
        out.println("Date: " + new Date());
        out.println("Content-Type: " + contentMimeType);
        out.println("Content-Length: " + fileLength);
        out.println(); // blank line between headers and content, very important !
        out.flush(); // flush character output stream buffer


        // rad file line by line and write to out

    }
    else{


    int fileLength = (int) file.length();
    String content = getContentType(fileRequested);

    // send HTTP Headers
        out.println("HTTP/1.1 200 OK");
        out.println("Date: " + new Date());
        out.println("Content-Type: " + content);
        out.println("Content-Length: " + fileLength);
        out.println(); // blank line between headers and content, very important !
        out.flush(); // flush character output stream buffer
     if (method.equals("GET")) { // GET method so we return content
```

93



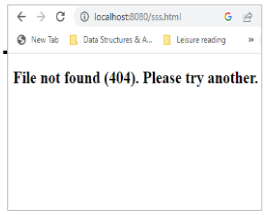

```
curl -I localhost:8080/
HTTP/1.1 200 OK
Content-Type: text/html
```

```
curl -I localhost:8080/working.html
HTTP/1.1 200 OK
Content-Type: text/html
```

```
curl localhost:8080/teaching.html
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <title>Document</title>
 </head>
 <body>
  <h2>This is the teaching page.</h2>

  I like to teach all kinds of CS courses, including
  <ul>
        <li>EECS1012</li>
        <li>EECS2030</li>
        <li>EECS2031</li>
        <li>EECS4413</li>
  </ul>
  Let me know which year you are in:
<select name="">
        <option value="" selected>first
        <option value="">second
        <option value="">third
        <option value="">fourth

  </select>

  <br><em>See you in class!</em>

 </body>
</html>
```

94

94




```
curl localhost:8080/xxx.html
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <title>Document</title>
 </head>
 <body>
  <h2>File not found (404). Please try another.</h2>
 </body>

</html>
```

```
curl -X POST localhost:8080
<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <title>Document</title>
 </head>
 <body>
  <h2>Method is not supported.<h2>
 </body>
</html>
```

```
curl -I -X POST localhost:8080
HTTP/1.1 501 Not Implemented
Content-Type: text/html
```

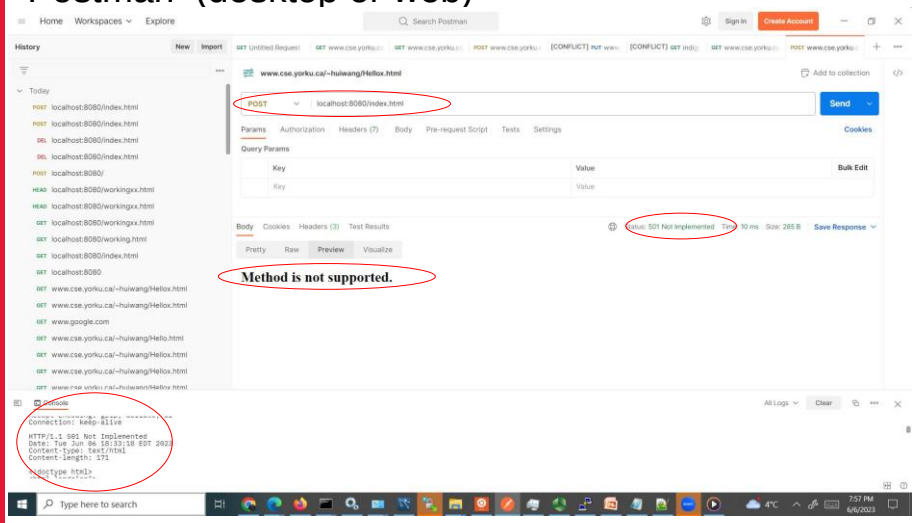
More on lab2 menu

Can also use postman, telnet,

95

95

# Postman (desktop or web)




Can see both html code and html page rendering.

96

96

# telnet

```
yu2031@RUSH-MO8A48ME44:~$ telnet localhost 8080
Trying ::1...
Connected to localhost.
Escape character is '^]'.
PUT / HTTP/1.1
HTTP/1.1 501 Not Implemented
Date: Tue Jun 06 18:36:05 EDT 2023
Content-type: text/html
Content-length: 171

<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <title>Document</title>
 </head>
 <body>
  <h2>Method is not supported.<h2>
 </body>
</html>
Connection closed by foreign host.
```

```
yu2031@RUSH-MO8A48ME44:~$ telnet localhost 8080
Trying ::1...
Connected to localhost.
Escape character is '^]'.
GET /xx.html HTTP/1.1
HTTP/1.1 404 File Not Found
Date: Tue Jun 06 18:36:32 EDT 2023
Content-type: text/html
Content-length: 189

<!doctype html>
<html lang="en">
 <head>
  <meta charset="UTF-8">
  <title>Document</title>
 </head>
 <body>
  <h2>File not found (404). Please try another.</h2>
 </body>
</html>
Connection closed by foreign host.
```

97

see the returned header + code

97

# Summary on Socket

- Generally, any message and process
- Keep on running
- Support multiple clients
- Can read and send (serializable) object,
- can access database

- As a HTTP client
  - send http request to website. Simulate browser

- As a HTTP server
  - handle http request, simulate as a website

- Class *InetAddress* are often used
  - *System.out.println( "socket connected at " + socket.getInetAddress());*

98

98