OTcl: Object-Oriented Tcl

- **OTcl: Object-Oriented Tcl**
  - **OTcl** is an **extension to Tcl** with **object-oriented programming**.
  - **Features of Otcl:**

    - **dynamically extensible**, (just like **Tcl**)
    - **builds on Tcl syntax and concepts**
    - compact yet powerful object programming system
    - fairly portable implementation (**2000 lines of C**)

- **OTcl and the network simulator NS**
  - The network simulator **NS** is actually a **special *version*** of a **OTcl programming language interpreter**
  - **The NS network simulation system:**

    - **NS** is **compiled** from a **OTcl language interpreter extended with** a ***built-in*** **library** of **network simulation classes**

  - The **NS2 system** has ***built-in*** **OTcl classes** that simulate:

    - **Routers** (node)
    - **Links**
    - **TCP** (Tahoe, Reno, Vegas, and many other flavors)
    - **FTP**
    - etc., etc.

  - **Example:** Execute this command in NS:

    ```
    Class info instances
    ```

    You will see a **long list of classes** that are **already defined** in **OTcl**:

    ```
    QueueMonitor/ED/Flow/TSW RandomVariable/ParetoII Queue/RED/Semantic
    Agent/rtProto/Dummy Est/ExpAvg Agent/IVS/Receiver Application/Worm
    Agent/TCP PacketHeader/ARP Agent/Message Agent/TCP/BayFullTcp
    Agent/SCTP/MultipleFastRtx Queue/dsRED/core PagePool/ProxyTrace/epa
    ConnTimer Trace/Enque Connector/AddSR PacketQueue HandoffManager/Sat
    RandomVariable/Normal Agent/TCP/Sack1 mrtObject Queue/dsRED/edge
    PagePool/EmpWebTraf Http/Server Application/Telnet Agent/UDP
    Agent/PGM QueueMonitor/ED/Flow Node/SatNode Classifier/Addr/MIPDecapsulator
    ```

```
SnoopQueue/In Application/Traffic/CBR_PP Channel/WirelessChannel PagePool
PacketHeader/GAF Classifier/Hash/Dest/Bcast SplitObject FSM/RenoAck
RtModule/Mcast PagePool/ProxyTrace Est/PointSample PktInTranMonitor
Application/DiffApp/RmstFilter Agent/LossMonitor/PLM PacketHeader/aSRM
RandomVariable/Constant Connector/RoutingHelper Queue/GK Phy/Repeater
.....
```

Notes:

- **Agent/TCP** is **TCP Tahoe**
- **Agent/UDP** is **UDP**

- Next, we will to learn how to:
  - **Create objects** in from a **built-in class** in **OTcl**
  - How to **invoke methods** in an **OTcl object**

- **Creating an object in OTcl**
  - The **most common way** to **create an object** is to use the `new` **command**:

```
new ClassName
```

The `new` **command returns** an **object handle (reference)**

  - **Example:** Create a **TCP Reno object**

```
set  myTCPAgent  [new Agent/TCP/Reno]
```

Note:

- The class `Agent/TCP/Reno` **simulates** the **TCP Reno protocol**.

- **Using an object in OTcl**
  - Just like **Java**, after you have created an object, you can **invoke methods** defined in that object using:

```
$OTclObjectVar   METHODNAME   ARG1   ARG2   ...
```

- **Accessing variables in an object**
  - The **`set` method** is used to **read/write variables** in an **OTcl object**
  - **Assign a value to a variable in an object:**

```
$OTclObjectVar  set  varName  VALUE
```

  - **Reading a value to a variable in an object:**

```
$OTclObjectVar  set  varName

Usually:
 set  $x  [ $OTclObjectVar  set  varName ]
```

- **What you need to learn in order to use NS**
  - Programming in NS does not happen in the OTcl language
  - All the network protocol modules (e.g., TCP protocols) have been implemented as **OTcl classes** in NS

    (So you don't need to write network modules (like TCP, FTP, etc) to simulate a network)

  - All you need to learn is:
    - The **name of the OTcl *class*** that implements the network module that you want to use
    - The **names of the *methods*** of the class

      The methods implement the behavior of the protocol modules.
    - The **names of the *variables*** of the class - setting of some variables can affect the behavior of the protocol module (e.g., packet size, queue length, etc)
  - Once you know these, you can use OTcl to create the modules you need and connect them

---

- **Further reading**
  - OTcl Tutorial: click here
