INTRODUCTION TO ANT

OVERVIEW

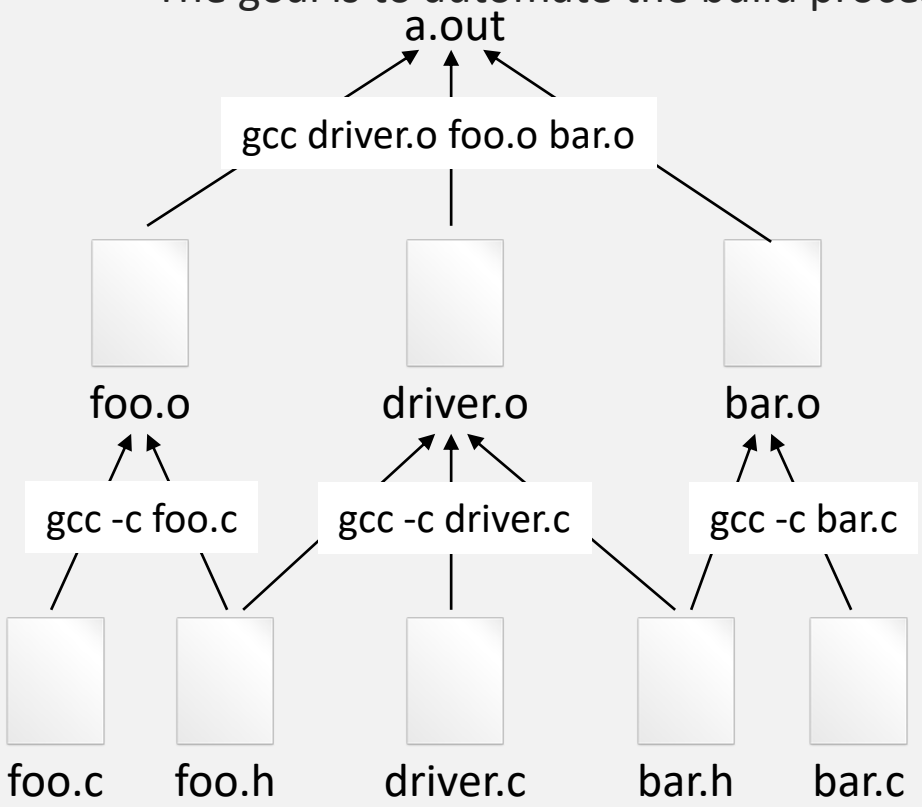
- What is Ant?
- Installing Ant
- Anatomy of a build file
  - Projects
  - Properties
  - Targets
  - Tasks
- Example build file
- Running a build file

WHAT IS ANT?

- Ant is a Java based tool for automating the build process
- Similar to make but implemented using Java
  - Platform independent commands (works on Windows, Mac & Unix)
- XML based format
  - Avoids the dreaded tab issue in make files
- Easily extendable using Java classes
- Ant is an open source (free) Apache project

AUTOMATING THE BUILD (C & MAKE)

- The goal is to automate the build process




```
a.out: driver.o foo.o bar.o
   gcc driver.o foo.o bar.o
driver.o: driver.c foo.h bar.h
   gcc -c driver.c
foo.o: foo.c foo.h
   gcc -c foo.c
bar.o:
   gcc -c bar.c
```

```
linux3[1]% make
gcc -c driver.c
gcc -c foo.c
gcc -c bar.c
gcc driver.o foo.o bar.o
linux3[2]%
```

INSTALLING ANT

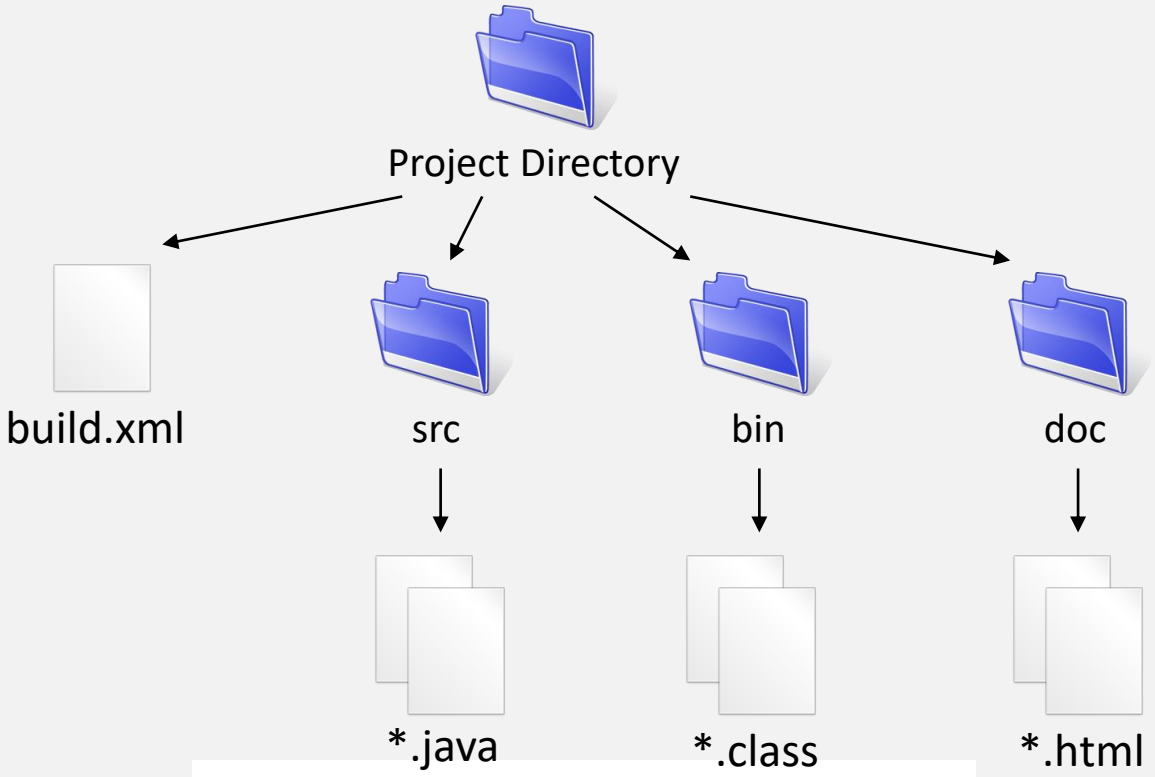
- Ant can be downloaded from…
  - http://ant.apache.org/
- Ant comes bundled as a zip file or a tarball
- Simply unwrap the file to some directory where you want to store the executables
  - I typically unwrap the zip file into C:\Program Files, and rename to C:\Program Files\ant\
  - This directory is known as ANT_HOME

ANT SETUP

- Set the ANT_HOME environment variable to where you installed Ant
- Add the ANT_HOME/bin directory to your path
- Set the JAVA_HOME environment variable to the location where you installed Java
- Setting environment variables
  - Windows: right click My Computer → Properties → Advanced → Environment Variables
  - UNIX: shell specific settings

PROJECT ORGANIZATION

- The following example assumes that your workspace will be organized like so…



ANATOMY OF A BUILD FILE

- Ant's build files are written in XML
  - Convention is to call file build.xml
- Each build file contains
  - A project
  - At least 1 target
- Targets are composed of some number of tasks
- Build files may also contain properties
  - Like macros in a make file
- Comments are within <!-- --> blocks

PROJECTS

- The project tag is used to define the project you wish to work with
- Projects tags typically contain 3 attributes
  - name – a logical name for the project
  - default – the default target to execute
  - basedir – the base directory for which all operations are done relative to
- Additionally, a description for the project can be specified from within the project tag

BUILD FILE

```
<project name="Sample Project" default="compile" basedir=".">

  <description>
    A sample build file for this project
  </description>

</project>
```

PROPERTIES

- Build files may contain constants (known as properties) to assign a value to a variable which can then be used throughout the project
  - Makes maintaining large build files more manageable
- Projects can have a set of properties
- Property tags consist of a name/value pair
  - Analogous to macros from make

BUILD FILE WITH PROPERTIES

```
<project name="Sample Project" default="compile" basedir=".">

  <description>
    A sample build file for this project
  </description>

  <!-- global properties for this build file -->
  <property name="source.dir" location="src"/>
  <property name="build.dir" location="bin"/>
  <property name="doc.dir" location="doc"/>

</project>
```

TARGETS

- The target tag has the following required attribute
  - name – the logical name for a target
- Targets may also have optional attributes such as
  - depends – a list of other target names for which this task is dependant upon, the specified task(s) get executed first
  - description – a description of what a target does
- Like make files, targets in Ant can depend on some number of other targets
  - For example, we might have a target to create a jarfile, which first depends upon another target to compile the code
- A build file may additionally specify a default target

BUILD FILE WITH TARGETS

```
<project name="Sample Project" default="compile" basedir=".">

  ...

  <!-- set up some directories used by this project -->
  <target name="init" description="setup project directories">
  </target>

  <!-- Compile the java code in src dir into build dir -->
  <target name="compile" depends="init" description="compile java sources">
  </target>

  <!-- Generate javadocs for current project into docs dir -->
  <target name="doc" depends="init" description="generate documentation">
  </target>

  <!-- Delete the build & doc directories and Emacs backup (*~) files -->
  <target name="clean" description="tidy up the workspace">
  </target>

</project>
```

TASKS

- A task represents an action that needs execution
- Tasks have a variable number of attributes which are task dependant
- There are a number of build-in tasks, most of which are things which you would typically do as part of a build process
  - Create a directory
  - Compile java source code
  - Run the javadoc tool over some files
  - Create a jar file from a set of files
  - Remove files/directories
  - And many, many others…
    - For a full list see: http://ant.apache.org/manual/coretasklist.html

INITIALIZATION TARGET & TASKS

- Our initialization target creates the build and documentation directories
  - The mkdir task creates a directory

```
<project name="Sample Project" default="compile" basedir=".">

 ...

 <!-- set up some directories used by this project -->
 <target name="init" description="setup project directories">
  <mkdir dir="${build.dir}"/>
  <mkdir dir="${doc.dir}"/>
 </target>

 ...

</project>
```

COMPILATION TARGET & TASKS

- Our compilation target will compile all java files in the source directory
  - The javac task compiles sources into classes
  - Note the dependence on the init task

```
<project name="Sample Project" default="compile" basedir=".">

 ...

 <!-- Compile the java code in ${src.dir} into ${build.dir} -->
 <target name="compile" depends="init" description="compile java sources">
  <javac srcdir="${source.dir}" destdir="${build.dir}"/>
 </target>

 ...

</project>
```

JAVADOC TARGET & TASKS

- Our documentation target will create the HTML documentation
  - The javadoc task generates HTML documentation for all sources

```
<project name="Sample Project" default="compile" basedir=".">

 ...

 <!-- Generate javadocs for current project into ${doc.dir} -->
 <target name="doc" depends="init" description="generate documentation">
  <javadoc sourcepath="${source.dir}" destdir="${doc.dir}"/>
 </target>

 ...

</project>
```

CLEANUP TARGET & TASKS

- We can also use ant to tidy up our workspace
  - The delete task removes files/directories from the file system

```
<project name="Sample Project" default="compile" basedir=".">
 ...
 <!-- Delete the build & doc directories and Emacs backup (*~) files -->
  <target name="clean" description="tidy up the workspace">
   <delete dir="${build.dir}"/>
   <delete dir="${doc.dir}"/>
   <delete>
    <fileset defaultexcludes="no" dir="${source.dir}" includes="**/*~"/>
   </delete>
  </target>
 ...
</project>
```

COMPLETED BUILD FILE (1 OF 2)

```
<project name="Sample Project" default="compile" basedir=".">

  <description>
    A sample build file for this project
  </description>

  <!-- global properties for this build file -->
  <property name="source.dir" location="src"/>
  <property name="build.dir" location="bin"/>
  <property name="doc.dir" location="doc"/>

  <!-- set up some directories used by this project -->
  <target name="init" description="setup project directories">
    <mkdir dir="${build.dir}"/>
    <mkdir dir="${doc.dir}"/>
  </target>
```

COMPLETED BUILD FILE (2 OF 2)

```
<!-- Compile the java code in ${src.dir} into ${build.dir} -->
 <target name="compile" depends="init" description="compile java sources">
   <javac srcdir="${source.dir}" destdir="${build.dir}"/>
 </target>
<!-- Generate javadocs for current project into ${doc.dir} -->
 <target name="doc" depends="init" description="generate documentation">
   <javadoc sourcepath="${source.dir}" destdir="${doc.dir}"/>
 </target>

 <!-- Delete the build & doc directories and Emacs backup (*~) files -->
 <target name="clean" description="tidy up the workspace">
   <delete dir="${build.dir}"/>
   <delete dir="${doc.dir}"/>
   <delete>
     <fileset defaultexcludes="no" dir="${source.dir}" includes="**/*~"/>
   </delete>
 </target>

</project>
```

RUNNING ANT – COMMAND LINE

- Simply cd into the directory with the build.xml file and type ant to run the project default target
- Or, type ant followed by the name of a target

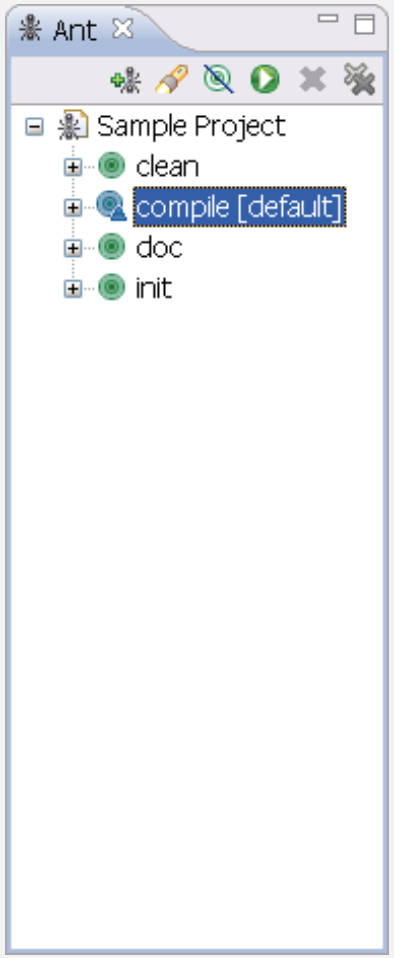
RUNNING ANT – COMMAND LINE

C:\WINDOWS\system32\cmd.exe

```
C:\Documents and Settings\Dan\workspace\hello-world>ant
Buildfile: build.xml

init:
    [mkdir] Created dir: C:\Documents and Settings\Dan\workspace\hello-world\bin

    [mkdir] Created dir: C:\Documents and Settings\Dan\workspace\hello-world\doc


compile:
    [javac] Compiling 1 source file to C:\Documents and Settings\Dan\workspace\h
ello-world\bin

BUILD SUCCESSFUL
Total time: 1 second
C:\Documents and Settings\Dan\workspace\hello-world>
```

RUNNING ANT – ECLIPSE

- Eclipse comes with out of the box support for Ant
  - No need to separately download and configure Ant
- Eclipse provides an Ant view
  - Window → Show View → Ant
- Simply drag and drop a build file into the Ant view, then double click the target to run



OTHER WAYS TO RUN???

- Build tools like Jenkins
- Automated background tasks...
- From Within Maven…..
- ……..

DIGGING DEEPER INTO ANT TASKS

WHAT ARE TASKS?

Ant has a large set of built-in tasks, such as:

`<echo ...>`	output a message
`<mkdir ...>`	create a directory (if it doesn't exist)
`<copy ...>`	copy a file, directory, or tree
`<javac ...>`	compile files using java compiler
`<jar ...>`	create a jar file
`<junit ...>`	run JUnit tests

<PROPERTY NAME="SRC" VALUE="...">

- Defines a variable ("property") that can be used throughout a build script.
- To access value of a property use: ${propertyname}.
- Useful for defining names of locations and files used repeatedly.

Example:

```
<property name="src.dir" value="src/java"/>
<javac ...>
  <src path="${src.dir}"/>
</javac>
```

<PROPERTY ...> (2)

- Read all the properties from a file. The file is a plain text file with lines of the form "name=value", like Java properties files

```
<property file="build.properties"/>
```

- Properties can be imported from system environment!
- Prefix environment properties with a "env."

```
<property environment="env"/>
<echo message=
        "CLASSPATH is ${env.CLASSPATH}"/>
<echo message=
        "JAVA_HOME is ${env.JAVA_HOME}"/>
```

<COPY FILE="*PATTERN*" TOFILE="..."/>

- Copies a file or set of files to another location.
- Does not overwrite existing files if they are newer than the source file (unless you specify that you want it to overwrite).

Copy a single file.

```
<copy file="${src.dir}/myfile.txt"
      tofile="${target.dir}/mycopy.txt"/>
```

<COPY TODIR="...">: COPY SETS OF FILES

- Copy files from one directory to another, omit any java source files.

```
<copy todir="${dest.dir}" >
    <fileset dir="src">
       <exclude name="**/*.java"/>
    </fileset>
</copy>
```

- Copy all files from the directory “../backup/” to “src_dir”. Replace occurrences of "@TITLE@" in the files with "Foo".

```
<copy todir="../backup">
    <fileset dir="src_dir"/>
    <filterset>
    <filter token="TITLE" value="Foo Bar"/>
    </filterset>
</copy>
```

<DELETE>

- Deletes files, directories, or sets of files.
- Delete a single file.

```
<delete file="/lib/ant.jar"/>
```

- Delete all *.bak files from this directory and sub-directories.

```
<delete>
 <fileset dir="." includes="**/*.bak"/>
</delete>
```

- Delete the build directory and everything in it.

```
<delete includeEmptyDirs="true">
    <fileset dir="build"/>
</delete>
```

<ECHO>

Display a message on terminal. It has 2 forms:

- Display a one-line message:

```
<echo message="Hello Ants" />
```

```
[echo] Hello Ants
```

- Display many lines of text:

```
<echo>
Usage:  ant target
clean  - delete compiler output files
build  - compile source code
dist   - create a distribution
</echo>
```

USING <ECHO>

A good `build.xml` file should have a `help` or `usage` target:

```
<project name="myapp" default="help">


<target name="help">
  <echo>
Usage:  ant target

where 'target' is one of:
  clean  - delete compiler output files
  build  - compile source code
  dist   - create a distribution
  </echo>
</target>
```

<MKDIR DIR="..."/>

- Create a directory.

```
<mkdir dir="${dist.dir}"/>
```

- Creates a subdirectory named "jars" in the location specified by the "`dist.dir`" property.

```
<mkdir dir="${dist.dir}/jars"/>
```

<JAVAC>

- Compiles Java source code.
- Attempts to analyze source such that up to date `.class` file are not recompiled.

Example: Compile all java source files under ${src.dir} and put the .class files in the ${build.classes} directory. Include debugging information in the .class files.

```
<javac srcdir="${src}"
    destdir="${build.classes}"
    classpath="mylib.jar"
    debug="true"/>
```

<JAVAC ...> (2)

- You can specify additional source directories and further restrict which files are compiled using `include` and `exclude`.

```
<javac destdir="${build}"
      classpath="xyz.jar" debug="on">
      <src path="${src}"/>
      <src path="${src2}"/>
      <include name="package/p1/**"/>
      <include name="package/p2/**"/>
      <exclude name="package/p1/test/**"/>
</javac>
```

<JAR ...>

- Creates a JAR file from a set of files or updates an existing JAR.
- Will automatically supply a manifest file for the JAR or use one you specify.

Example: make a jar file including all files in build/classes

```
<jar jarfile="${dist}/lib/myapp.jar"
     basedir="${build}/classes"/>
```

<JAR ...>

- Create a JAR file from all the files in ${build}/classes and ${src}/resources. (two sets of files)
- Any files named `*Test.class` in the build directory are not included in the JAR.

```
<jar jarfile="${dist}/lib/myapp.jar">
    <fileset dir="${build}/classes"
       excludes="**/*Test.class" />
    <fileset dir="${src}/resources"/>
</jar>
```

<JAVADOC>

- Creates Javadocs from Java source code files.

Example: Build Javadoc only for the packages beginning with "`org.ske...`" in the ${`src`}\ directory.

```
<javadoc packagenames="org.ske.*"
         sourcepath="${src}"
         destdir="${doc}/api"/>
```

This command will search all subdirectories of ${src} for *.java files.

<JAVA>

- Invoke a Java program from within an Ant build file.
- Can fork a separate process so that a System.exit() does not kill the Ant build.

```
<java classname="test.Main">
  <arg value="some-arg-to-main"/>
  <classpath>
    <pathelement location="test.jar"/>
    <pathelement
        path="${java.class.path}"/>
  </classpath>
</java>
```

<JAVA>

Invoke a class named test.Main in a separate Java VM. The Java VM is invoked using the options:

-Xrunhprof:cpu=samples,file=log.txt,depth=3

to request profiling.

```
<java classname="test.Main" fork="yes">
  <sysproperty key="DEBUG" value="true"/>
  <arg value="-h"/>
  <jvmarg value=
   "-Xrunhprof:cpu=samples,file=log.txt,depth=3"/>
</java>
```

MORE ANT TASKS

- The Apache Ant distribution includes more than 50 **core** tasks and many **optional** tasks.
- Examples: zip, gzip, war (create a war file),
- Many tasks correspond to standard Linux commands, like mkdir, copy, move.
- You can write your own Ant tasks using <taskdef />.
- See Ant manual (ant/docs directory) for how to use each task.

TOOLS

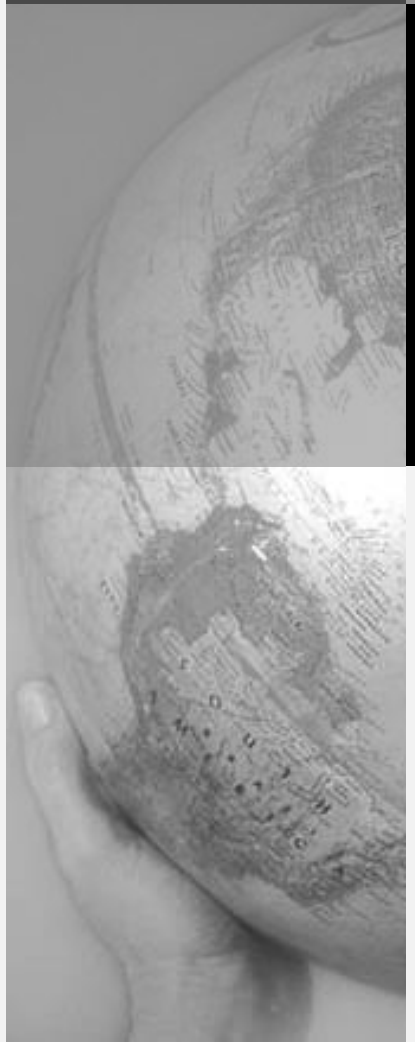
- List of Ant tools:

  `http://ant.apache.org/external.html`

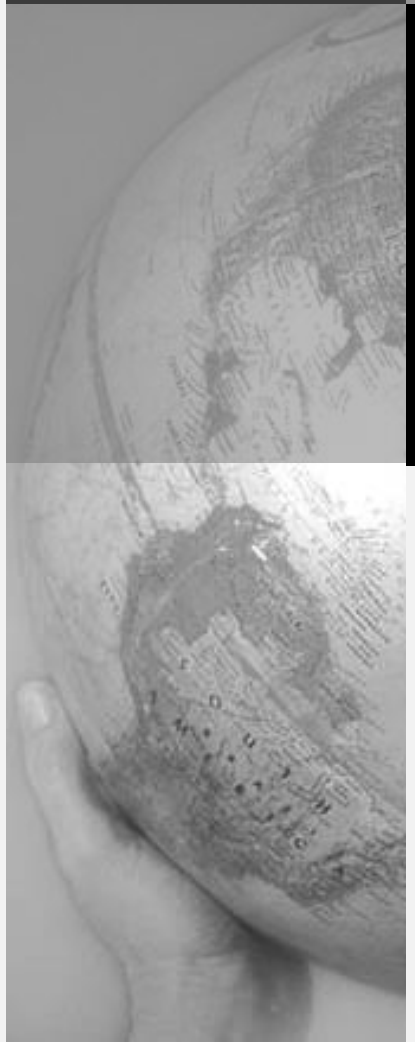
- NetBeans creates build.xml files for NetBeans projects.
- Eclipse can "export" an Ant build file, but it contains a lot of Eclipse-specific references that make the build file not portable.
- Ivy (`http://ant.apache.org/ivy`) is a dependency manager for Ant. Automatically downloads dependencies, similar to what Maven does (Ivy can use Maven repositories).

RESOURCES

- Ant Home: **`http://ant.apache.org`**
- *Apache Ant Manual*. Installed with ant, in the **`ant/docs`** directory. The *Ant Manual* documents all Ant tasks.
- *Ant: The Definitive Guide*. O'Reilly. Terse, but lots of info.



Questions and Answers.



Thanks for listening!!!

Seshagiri Sriram
Email: SeshagiriSriram@gmail.com