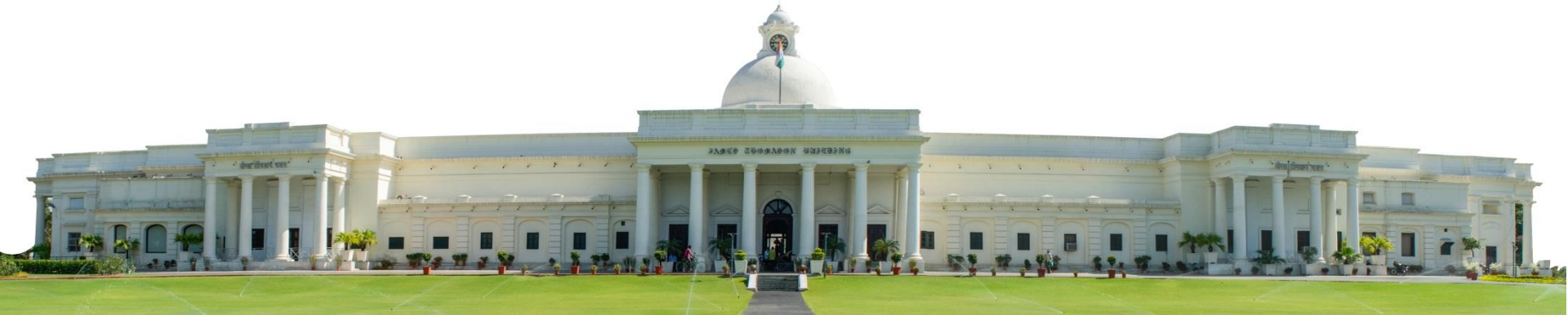
CSN-103: Fundamentals of Object Oriented Programming

Instructor: Dr. Rahul Thakur

Assistant Professor, Computer Science and Engineering, IIT Roorkee



Introduction

- All example classes were taken from the same name space
- This means that a unique name had to be used for each class to avoid name collisions
- Java provides a mechanism for partitioning the class name space into more manageable chunks → **Package**
- The package is both a **naming** and a **visibility control** mechanism
- You can define classes inside a package that are **not** accessible by code outside that package
- You can also define class members that are exposed only to other members of the same package

Defining a Package

- To create a package
  - Include a **package** command as the first statement in a Java source file
  - Any classes declared within that file will belong to the specified package
- If you omit the **package** statement
  - The classes are put into the default package, which has no name
  - Most of the time, it is best to define a package for your code

Package

- Java uses file system directories to store packages
- The **.class** files for the classes belonging to a package must be stored in a directory whose name is same as the package
- The directory name must match the package name exactly
- More than one file can include the same **package**
- The **package** statement simply specifies to which package the classes defined in a file belong

Package Hierarchy

- You can create a hierarchy of packages
  - By simply separating each package name from the one above it by use of a period

```
package pkg1[.pkg2[.pkg3]];
```

  - Need to be stored in pkg1\pkg2\pkg3 in a Windows environment

Finding Packages

- Packages are mirrored by directories
- How does the Java run-time system know where to look for packages that you create?
- Three ways:
  - By default, the Java run-time system uses the current working directory
  - You can use the **–classpath/-cp** option with **java** and **javac** to specify the path to your classes
  - You can specify a directory path or paths by setting the **CLASSPATH** environmental variable

Compilation and Execution

- The easiest way:
  - Simply create the package directories below your current development directory
  - Put the **.class** files into the appropriate directories (packages)
  - Then execute the programs from the development directory

Compilation and Execution

- Create and put file **AccountBalance.java** in a directory called **MyPack**
- Compile the file
  - Make sure that the resulting **.class** file is also in the **MyPack** directory
  - Execute the **AccountBalance** class:

    java MyPack.AccountBalance
- You will need to be in the directory above **MyPack** when you execute this command
  - OR use –classpath / -cp at the time of execution

Access Protection

- Classes and packages: Means of encapsulating, Acts as containers
- Classes act as containers for data and code
- Packages act as containers for classes and other subordinate packages
- Four categories of visibility for class members:
  - Subclasses in the same package
  - Non-subclasses in the same package
  - Classes that are neither in the same package nor subclasses
  - Subclasses in different packages

Access Protection

- Anything declared **public** can be accessed from anywhere
- Anything declared **private** cannot be seen outside of its class
- When a member does not have an **explicit access** specification, it is visible to subclasses as well as to other classes in the same package → This is the default access
- If you want to allow an element to be seen outside your current package, but only to classes that subclass your class directly
  - Declare that element **protected**

Class Member Access

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

import Statement

- Packages are a good mechanism for compartmentalizing diverse classes
- All of the built-in Java classes are stored in packages
  - All of the standard classes are stored in some named package
- Classes within packages must be fully qualified with their package name
  - Tedious to type in the long dot-separated package path name for every class you want to use
- **import** statement
  - Bring certain classes, or entire packages, into visibility

import Statement

- **import** statements occur
  - Immediately following the **package** statement (if it exists)
  - And before any class definitions
- This is the general form of the **import** statement:

import *pkg1* [.*pkg2*].(*classname* | *);

- All of the standard Java classes included with Java are stored in a package called **java**

Examples:

```
import java.util.Date;
import java.io.*;
import java.lang.*;
```

import and Public

- When a package is imported
  - Items within the package declared as **public** will be available to non-subclasses in the importing code
- **For example: TestBalance.java**
  - **Balance** class is **public**
  - Also, its constructor and **show( )** method are **public**
  - They can be accessed by any type of code outside the **MyPack** package