OOP class
final with inheritance, Package Intro

Using final with Inheritance

- The keyword **final has three uses.**
- **First, it can be used to create the equivalent of a named** constant. This use was described in the preceding chapter.
- The other two uses of **final apply** to inheritance. Both are examined here.

final keyword with inheritance

- In inheritance, we must declare methods with final keyword for which we required to follow the same implementation throughout all the derived classes.
- Note that it is not necessary to declare final methods in the initial stage of inheritance(base class always).
- We can declare final method in any subclass for which we want that if any other class extends this subclass, then it must follow same implementation of the method as in the that subclass.

Using final to Prevent Overriding

- While method overriding is one of Java's most powerful features, there will be times when you will want to prevent it from occurring.
- To disallow a method from being overridden, specify **final as a modifier at the start of its declaration.**
- **Methods declared as final cannot** be overridden. The following fragment illustrates **final:**

```
class A {
  final void meth() {
    System.out.println("This is a final method.");
  }
}

class B extends A {
  void meth() { // ERROR! Can't override.
    System.out.println("Illegal!");
  }
}
```

Because **meth( ) is declared as final, it cannot be overridden in B. If you attempt to do**
so, a compile-time error will result.

- Methods declared as **final can sometimes provide a performance enhancement: The** compiler is free to *inline calls to them because it "knows" they will not be overridden by a* subclass.
- When a small **final method is called, often the Java compiler can copy the bytecode** for the subroutine directly inline with the compiled code of the calling method, thus eliminating the costly overhead associated with a method call.
- Inlining is an option only with **final methods.**
- **Normally, Java resolves calls to methods dynamically, at run time. This is called** *late binding. However, since* ***final methods cannot be overridden, a call to one can be resolved*** at compile time. This is called early binding.

Using final to Prevent Inheritance

- Sometimes you will want to prevent a class from being inherited. To do this, precede the class declaration with **final.**
- Declaring a class as final implicitly declares all of its methods as final, too.
- it is illegal to declare a class as both abstract and final since an abstract class is incomplete by itself and relies upon its subclasses to provide complete implementations.

Here is an example of a **final class:**

```
final class A {
  //...
}

// The following class is illegal.
class B extends A { // ERROR! Can't subclass A
  //...
}
```

As the comments imply, it is illegal for **B to inherit A since A is declared as final.**

```
class Demo{
   public final void demoMethod(){
      System.out.println("Hello");
   }
}
public class Sample extends Demo{
   public final void demoMethod(){
      System.out.println("demo method");
   }
   public static void main(String args[]){
      Sample obj = new Sample();
      obj.demoMethod();
   }
}
```

Output

```
C:\Sample>javac Sample.java
Sample.java:8: error: demoMethod() in Sample cannot override demoMethod() in Demo
public final void demoMethod(){
                  ^
overridden method is final
1 error
```

The Object Class

- There is one special class, Object, defined by Java.
- All other classes are subclasses of Object.
- Object is a superclass of all other classes.
- This means that a reference variable of type Object can refer to an object of any other class.
- Also, since arrays are implemented as classes, a variable of type Object can also refer to any array.
- Object defines the following methods, which means that they are available in every object.

Method	Purpose
Object clone( )	Creates a new object that is the same as the object being cloned.
boolean equals(Object *object*)	Determines whether one object is equal to another.
void finalize( )	Called before an unused object is recycled.
Class<?> getClass( )	Obtains the class of an object at run time.
int hashCode( )	Returns the hash code associated with the invoking object.
void notify( )	Resumes execution of a thread waiting on the invoking object.
void notifyAll( )	Resumes execution of all threads waiting on the invoking object.
String toString( )	Returns a string that describes the object.
void wait( ) void wait(long *milliseconds*) void wait(long *milliseconds*, int *nanoseconds*)	Waits on another thread of execution.

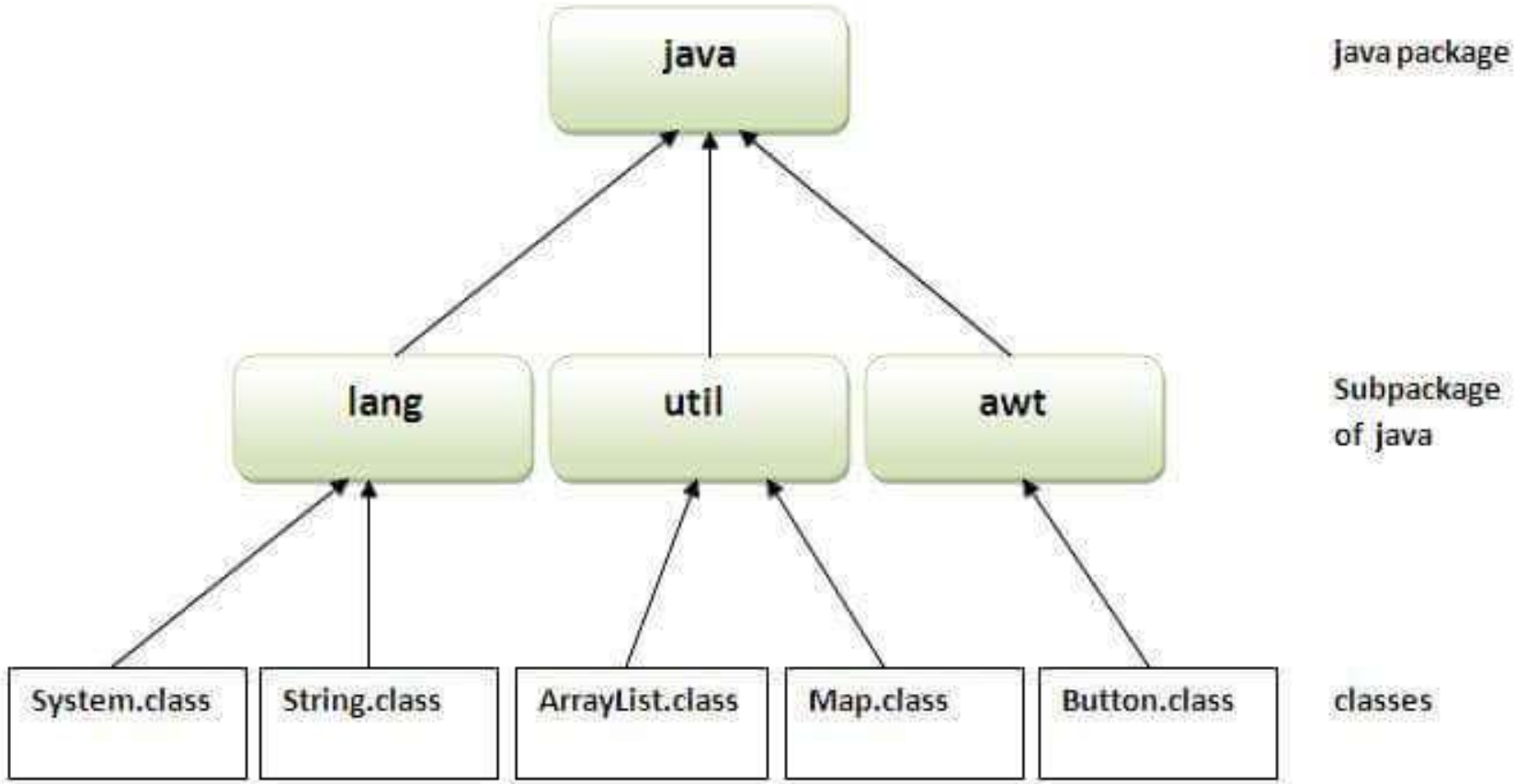
Packages

- A **java package** is a group of similar types of classes, interfaces and sub-packages.
- Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, etc.
- Package in java can be categorized in two form, built-in package and user-defined package.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision

- Some of the existing packages in Java are −
- **java.lang** − bundles the fundamental classes
- **java.io** − classes for input , output functions are bundled in this package


java
lang
util
awt
System.class
String.class
ArrayList.class
Map.class
Button.class
Java package
Subpackage
of java
classes

Simple example of java package

- The **package keyword** is used to create a package in java.

```
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
   }
}
```

How to compile java package

If you are not using any IDE, you need to follow the **syntax** given below:

javac -d directory javafilename

javac -d . Simple.java

Note:
The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc.

If you want to keep the package within the same directory, you can use . (dot).

How to run java package program

To Compile: javac -d . Simple.java

To Run: java mypack.Simple

Output:
Welcome to package