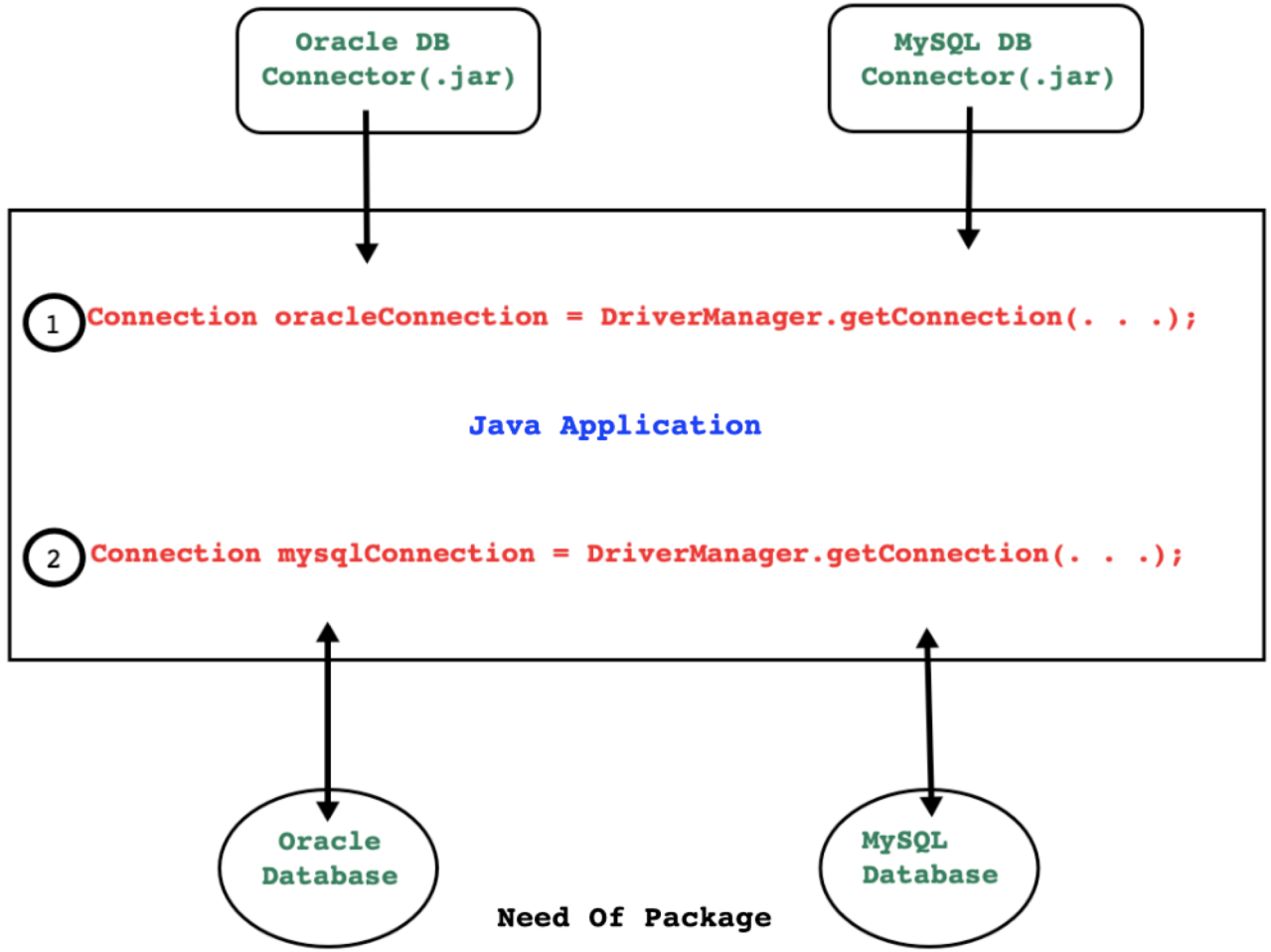
Object Oriented Programming with Java

Sandeep Kulange

Package


Oracle DB
Connector(.jar)
MySQL DB
Connector(.jar)
1 Connection oracleConnection = DriverManager.getConnection(. . .);
Java Application
2 Connection mysqlConnection = DriverManager.getConnection(. . .);
Oracle
Database
MySQL
Database
Need Of Package

Package

- Package is a Java language feature which helps developer to:
  1. To group functionally equivalent or related types together.
  2. To avoid naming clashing/collision/conflict/ambiguity in source code.
  3. To control the access to types.
  4. To make types easier to find(from the perspective of java docs).

- Consider following class:
  - java.lang.Object
    - Here java is main package, lang is sub package and Object is type name.

Package

- Not necessarily but as shown below, package can contain some or types.
    1. Sub package
    2. Interface
    3. Class
    4. Enum
    5. Exception
    6. Error
    7. Annotation Type

Package Creation

- package is a keyword in Java.
- To define type inside package, it is mandatory write package declaration statement inside .java file.
- Package declaration statement must be first statement inside .
- If we define any type inside package then it is called as packaged type otherwise it will be unpackaged type.
- Any type can be member of single package only.

```
package p1;   //OK
class Program{
    //TODO
}
```

```
package p1, p2; //NOT OK
class Program{
    //TODO
}
```

```
package p1; //OK
package p2; //NOT OK
class Program{
    //TODO
}
package p3; //Not OK
```

Un-named Package

- If we define any type without package then it is considered as member of unnamed/default package.
- Unnamed packages are provided by the Java SE platform principally for convenience when developing small or temporary applications or when just beginning development.
- An unnamed package cannot have sub packages.
- In following code, class Program is a part of unnamed package.

```
class Program{
    public static void main(String[] args) {
        System.out.println("Hello");
    }
}
```

Naming Convention

- For small programs and casual development, a package can be unnamed or have a simple name, but if code is to be widely distributed, unique package names should be chosen using qualified names.
- Generally Package names are written in all lower case to avoid conflict with the names of classes or interfaces.
- Companies use their reserved internet domain name to begin their package names. For example : com.example.mypackage
- Following examples will help you in deciding name of package:
    1. java.lang.reflect.Proxy
    2. oracle.jdbc.driver.OracleDriver
    3. com.mysql.jdbc.cj.Driver
    4. org.cdac.sunbeam.dac.utils.Date

How to use package members in different package?

- If we want to use types declared inside package anywhere outside the package then
    1. Either we should use fully qualified type name or
    2. import statement.
- If we are going to use any type infrequently then we should use fully qualified name.
- Let us see how to use type using package name.

```
class Program{
    public static void main(String[] args) {
        java.util.Scanner sc = new java.util.Scanner( System.in );
    }
}
```

How to use package members in different package?

- If we are going to use any type frequently then we should use import statement.
- Let us see how to import Scanner.

```
import java.util.Scanner;
class Program{
    public static void main(String[] args) {
        Scanner sc = new Scanner( System.in );
    }
}
```

How to use package members in different package?

- There can be be any number of import statements after package declaration statement
- With the help of(*) we can import entire package.

```
import java.util.*;
class Program{
    public static void main(String[] args) {
        Scanner sc = new Scanner( System.in );
    }
}
```

How to use package members in different package?

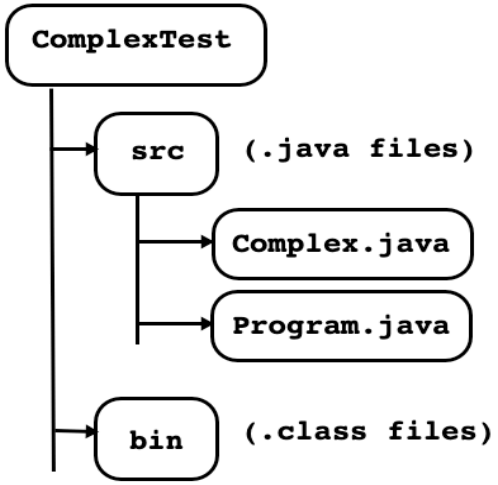
- Another, less common form of import allows us to import the public nested classes of an enclosing class. Consider following code.

```
import java.lang.Thread.State;
class Program{
    public static void main(String[] args) {
        Thread thread = Thread.currentThread( );
        State state = thread.getState( );
    }
}
```

- Note : java.lang package contains fundamental types of core java. This package is by default imported in every .java file hence to use type declared in java.lang package, import statement is optional.

Example 1

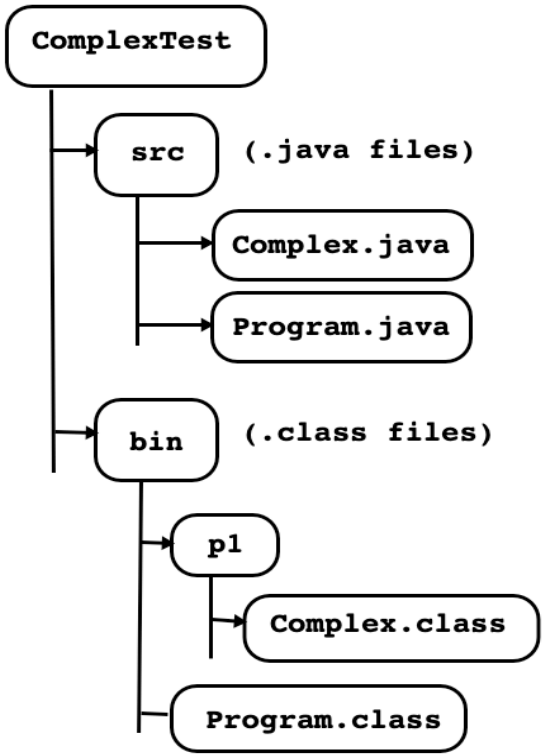
[Compile from here]



[Before Compilation]

1. Set path(if not set)
2. Compile "Complex.java"
3. set classpath
4. Compile "Program.java"
5. execute "Program.class"

[execute from here]



[After Compilation]

Example 1

```java
//Location : ./src/Complex.java
package p1;
class TComplex{
    @Override
    public String toString( ){
        return "TComplex.toString()";
    }
}
```

```java
//Location : ./src/Program.java
import p1.TComplex;
class Program{
    public static void main(String[] args) {
        //p1.TComplex c1 = new p1.TComplex( );  //or
        TComplex c1 = new TComplex( );
        System.out.println( c1.toString( ) );
    }
}
```

```
javac -d ./bin ./src/Complex.java   //output : p1/TComplex.class
export CLASSPATH=./bin
javac -d ./bin ./src/Program.java   //Output : Error
//TComplex is not public in p1; can not be accessed from outside package
```

Example 1

- Package name is physically mapped to the folder.
- Point to Remember : default access modifier of any type is package level private which is also called as default.

```
//location  :   ./src/Complex.java
??? class TComplex{ //here ??? means package level  private / default
    //TODO
}
```

- If we want to use any type inside same package as well as in different package then access modifier of type must be public.
- Access modifier of type(class/interface) can be either package level private/public only. In other words, type can not be private or protected.

```
//location  :   ./src/Complex.java
public class TComplex{
}
```

Example 1

- According to Java Language Specification(JLS), name of public type and name of .java file must be same. It means that, .java file can contain multiple non public types but only one public type.

```
//location  :   ./src/Complex.java
public class Complex{   //Now OK
     @Override
    public String toString( ){
        return "Complex.toString()";
    }
}
```

- Let us recompile above code:
  - javac -d ./bin ./src/Complex.java //output : p1/Complex.class
  - export CLASSPATH=./bin
  - javac -d ./bin ./src/Program.java //Output : Program.class
  - java Program //Output : Complex.toString()
- **Conclusion : It is possible to use packaged type from unpackaged type.**

Example 2

```
//Location : ./src/Complex.java
public class Complex{ //Unpackaged Type
    @Override
    public String toString( ){
        return "Complex.toString()";
    }
}
```

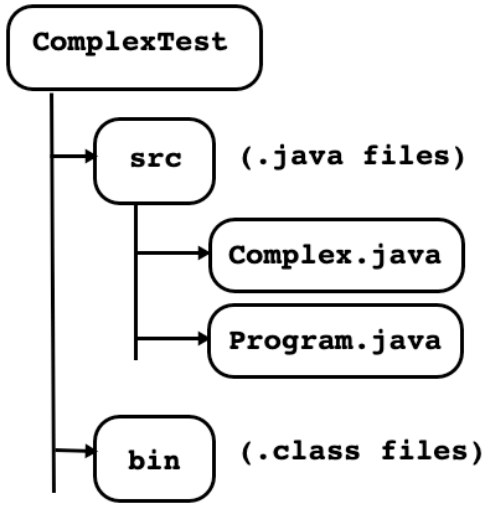
```
//Location : ./src/Program.java
package p1;
public class Program{
    public static void main(String[] args) {
        Complex c1 = new Complex( );
        System.out.println( c1.toString( ) );
    }
}
```

```
javac -d ./bin ./src/Complex.java    //OK:Complex.class
export CLASSPATH=./bin
javac -d ./bin ./src/Program.java    //error : cannot find symbol
```

- If we define any type without package then it is considered as a member of default package.
- Conclusion : Since we can not import default package, it is not possible to use unpackaged type from packaged type.

Example 3

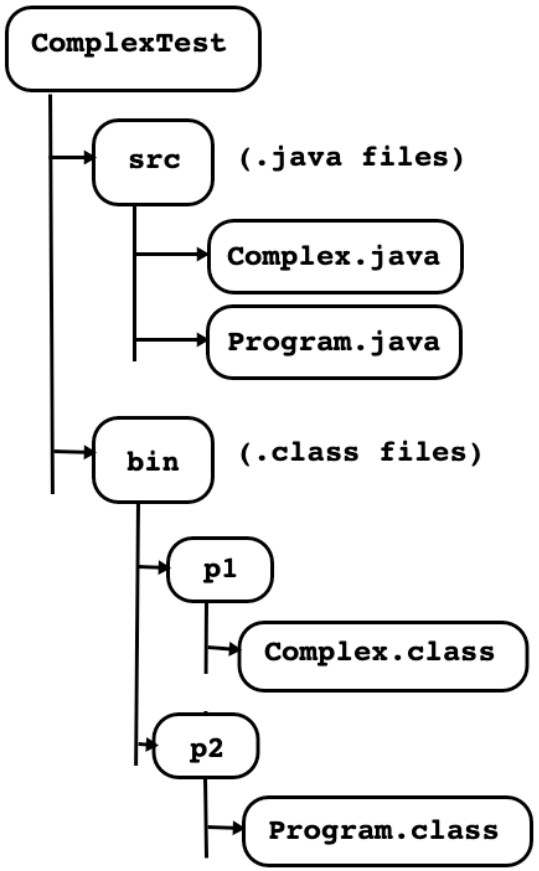
[Compile from here]



[Before Compilation]

1. Set path(if not set)
2. Compile "Complex.java"
3. set classpath
4. Compile "Program.java"
5. execute "Program.class"

[execute from here]



[After Compilation]

Example 3

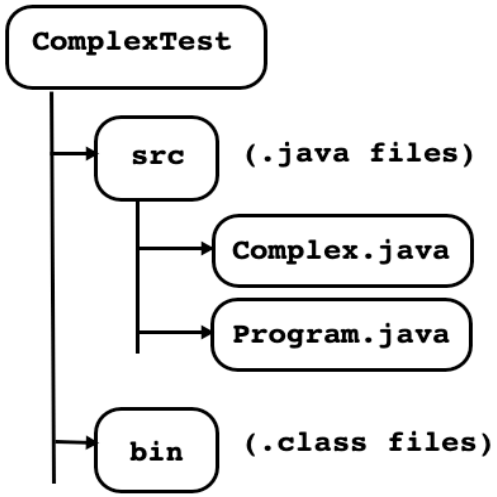
```java
//Location : ./src/Complex.java
package p1;
public class Complex{
    @Override
    public String toString( ){
        return "Complex.toString()";
    }
}
```

```java
//Location : ./src/Program.java
package p2;
import p1.Complex;
public class Program{
    public static void main(String[] args) {
        Complex c1 = new Complex( );
        System.out.println( c1.toString( ) );
    }
}
```

```
javac -d ./bin ./src/Complex.java    //OK:p1/Complex.class
export CLASSPATH=./bin
javac -d ./bin ./src/Program.java    //OK:p2/Program.class
//java Program  //Error
java p2.Program //Complex.toString()
```

Example 4

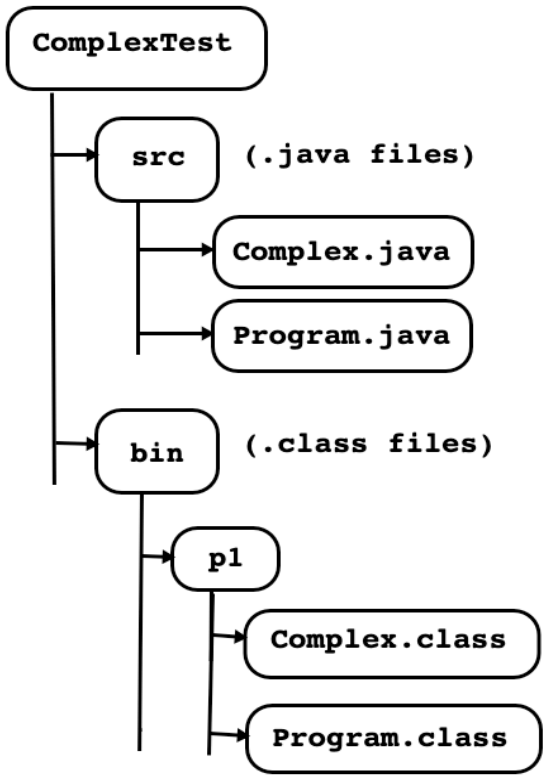
[Compile from here]



[Before Compilation]

1. Set path(if not set)
2. Compile “Complex.java”
3. set classpath
4. Compile “Program.java”
5. execute “Program.class”

[execute from here]



[After Compilation]

Example 4

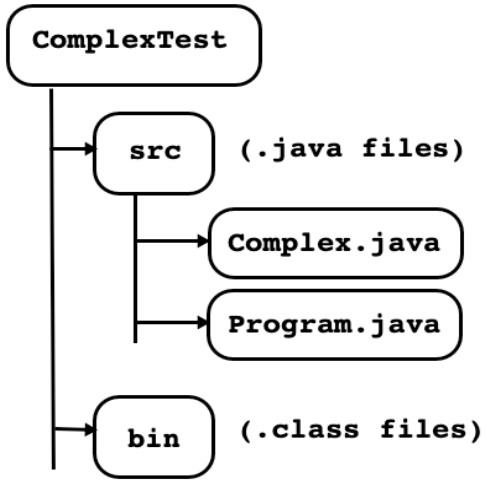
```java
//Location : ./src/Complex.java
package p1;
public class Complex{
    @Override
    public String toString( ){
        return "Complex.toString()";
    }
}
```

```java
//Location : ./src/Program.java
package p1;
import p1.Complex;  //Optional
public class Program{
    public static void main(String[] args) {
        Complex c1 = new Complex( );
        System.out.println( c1.toString( ) );
    }
}
```

```
javac -d ./bin ./src/Complex.java    //OK:p1/Complex.class
export CLASSPATH=./bin
javac -d ./bin ./src/Program.java    //OK:p1/Program.class
java p1.Program //Complex.toString()
```

Example 5

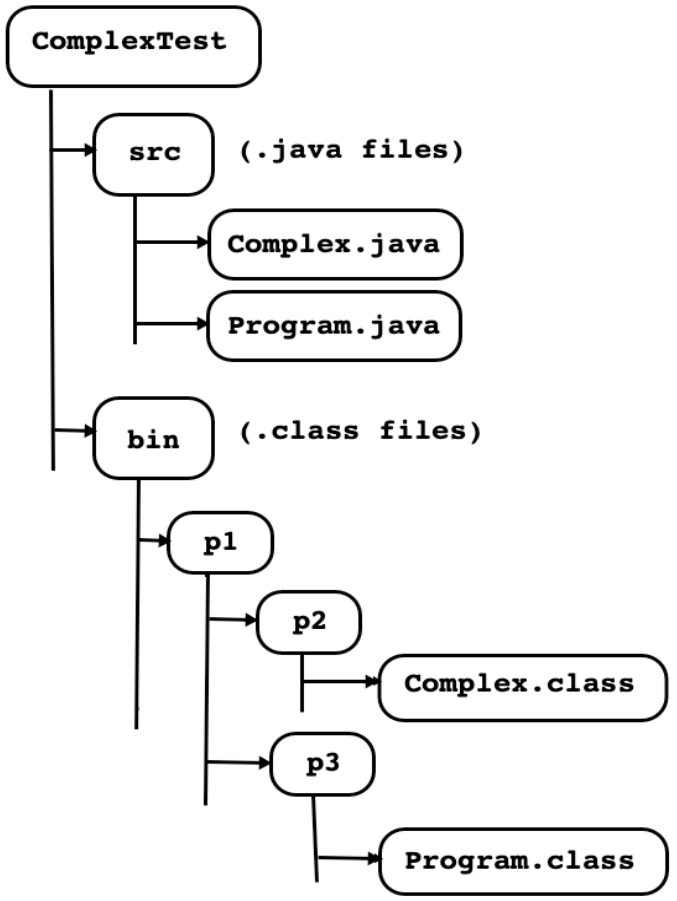
[Compile from here]



[Before Compilation]

1. Set path(if not set)
2. Compile “Complex.java”
3. set classpath
4. Compile “Program.java”
5. execute “Program.class”

[execute from here]



[After Compilation]

Example 5

```java
//Location : ./src/Complex.java
package p1.p2;
public class Complex{
    @Override
    public String toString( ){
        return "Complex.toString()";
    }
}
```

```java
//Location : ./src/Program.java
package p1.p3;
import p1.p2.Complex;
public class Program{
    public static void main(String[] args) {
        Complex c1 = new Complex( );
        System.out.println( c1.toString( ) );
    }
}
```

```
javac -d ./bin ./src/Complex.java    //OK:p1/p2/Complex.class
export CLASSPATH=./bin
javac -d ./bin ./src/Program.java    //OK:p1/p3/Program.class
java p1.p3.Program //Complex.toString()
```

Static Import

- If static members belonging to the same class then use of type name and dot operator is optional.

```
package p1;
public class Program{
    private static int number = 10;
    public static void main(String[] args) {
        System.out.println("Number  :   "+Program.number);  //OK    :   10
        System.out.println("Number  :   "+number);  //OK    :   10
    }
}
```

Static Import

- If static members belonging to the different class then use of type name and dot operator is mandatory.
- PI and pow are static members of java.lang.Math class. To use Math class import statement is not required.
- Consider Following code:

```
package p1;
public class Program{
    public static void main(String[] args) {
        float radius = 10.5f;
        float area = ( float )( Math.PI * Math.pow( radius, 2 );
        System.out.println( "Area    :    "+area );
    }
}
```

Static Import

- There are situations where you need frequent access to static final fields (constants) and static methods from one or two classes. Prefixing the name of these classes over and over can result in cluttered code. The static import statement gives you a way to import the constants and static methods that you want to use so that you do not need to prefix the name of their class.
- Consider code:

```
package p1;
import static java.lang.System.out;
import static java.lang.Math.*;
public class Program{
    public static void main(String[] args) {
        float radius = 10.5f;
        float area = ( float )( PI * pow( radius, 2 );
        out.println( "Area    :    "+area );
    }
}
```

Thank You.

[sandeepkulange@sunbeaminfo.com]