

```
javac -cp . `find aaa -name *.java`  
jar tf ggc-skeleton.jar view  
jar xf ggc-skeleton.jar export  
jar cfv a.jar aaa
```

Organizing Classes in Java

Packages

Organization of Code - Motivation

- It is important to group together classes/interfaces that are related
- We need to provide a layer of access / protection
- Avoid name clash

Solution in Java: Packages

- Classes can be grouped in a collection called *package*
 - Each package has a name
 - A package can also have sub-packages
 - Java's standard library consists of hierarchical packages, such as `java.lang` and `java.util`
- Definition of the ***package-private*** access level
 - The class can be *package-private*
 - Members of a class can be *package-private*
- Packages provides a name space
 - Classes have a simple name (just the name of the class) and
 - A full name: concatenation of the package name and simple name
 - The name of a class **is always** the full name
 - Classes with same (**simple**) name can be encapsulated in different packages

Sub-packages

- We can create hierarchies of nested packages
- Sub-package name must always be prefixed with name of parent package (separated by '.')
 - `java.awt` and `java.awt.event`
- *Sub-packages* have a similar name to parent package but are not actually contained inside
 - `java.awt` represents all classes belonging to this package only
 - **does not** consider classes of sub-packages of `java.awt`
 - **does not** contain `java.awt.event`

Class importation (1)

- Two ways of accessing **PUBLIC** classes of another package
 - 1) explicitly give the full package name before the class name.
 - E.g.

```
java.util.Date today = new java.util.Date( );
```
 - 2) import the package by using the `import` statement at the top of your source files (but below package statements). No need to give package name any more.
 - to import a single class from the `java.util` package

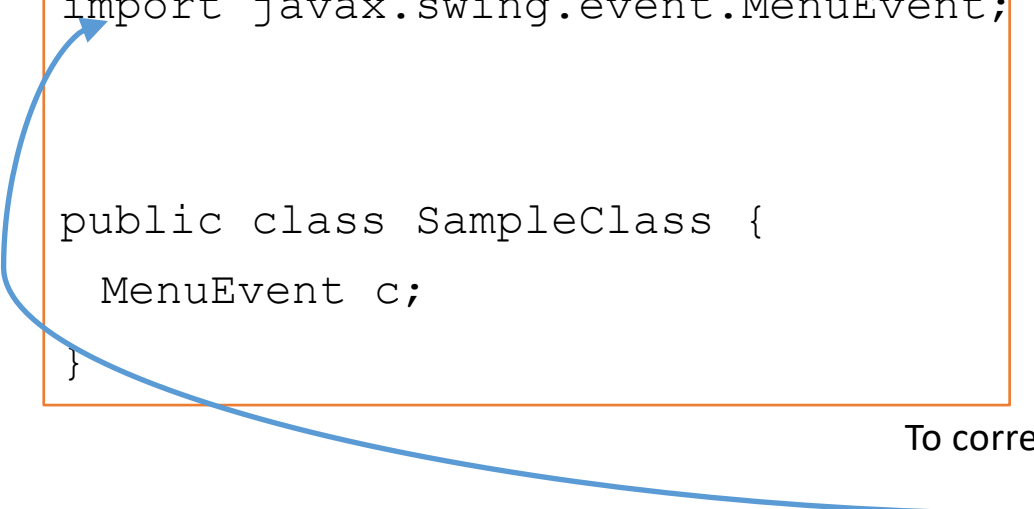
```
import java.util.Date;  
Date today = new Date( );
```
 - to import all the public classes from the `java.util` package

```
import java.util.*;  
Date today = new Date( );
```
 - `*` is used to import classes at the current package level. It will **NOT** import classes in a sub-package.
- A class can access other classes belonging to the same package using just the simple name
- All classes of the `java.lang` package are imported automatically into all classes

Class importation - Example

Consider the class javax.swing.event.MenuEvent

```
import javax.swing.*;  
import javax.swing.event.MenuEvent;  
  
public class SampleClass {  
    MenuEvent c;  
}
```



```
%> javac SampleClass.java
```

```
SampleClass.java:4: cannot find symbol  
Symbol   : class MenuEvent  
Location: class SampleClass  
    MenuEvent c;  
    ^  
1 error
```

To correct this error you need to add:

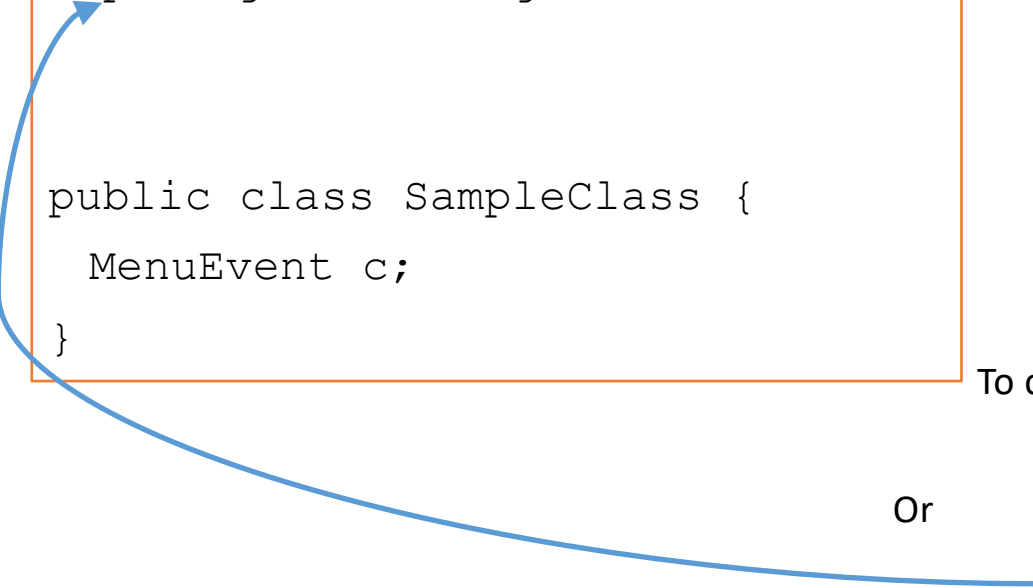
```
import javax.swing.event.MenuEvent;
```

Or

Class importation - Example

Consider the class `javax.swing.event.MenuEvent`

```
import javax.swing.*;  
import javax.swing.event.*;  
  
public class SampleClass {  
    MenuEvent c;  
}
```



```
%> javac SampleClass.java
```

```
SampleClass.java:4: cannot find symbol  
Symbol   : class MenuEvent  
Location: class SampleClass  
    MenuEvent c;  
      ^  
1 error
```

To correct this error you need to add:

```
import javax.swing.event.MenuEvent;
```

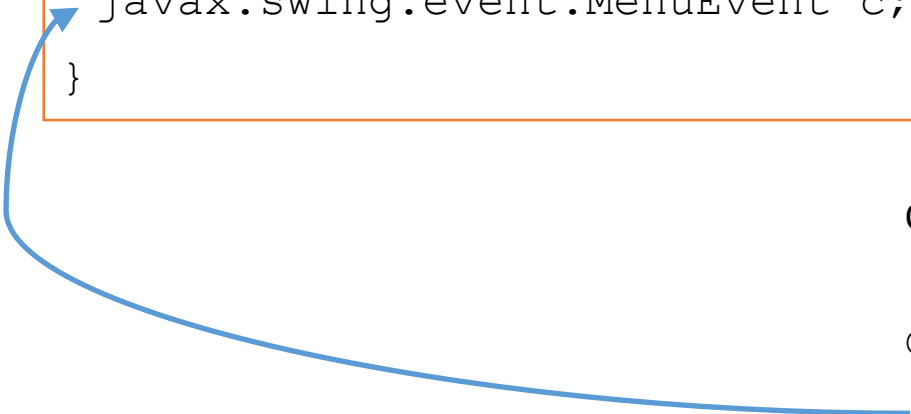
Or

```
import javax.swing.event.*;
```

Class importation - Example

Consider the class javax.swing.event.MenuEvent

```
import javax.swing.*;  
  
public class SampleClass {  
    javax.swing.event.MenuEvent c;  
}
```



```
%> javac SampleClass.java
```

```
SampleClass.java:4: cannot find symbol  
Symbol   : class MenuEvent  
Location: class SampleClass  
    MenuEvent c;  
    ^  
1 error
```

To correct this error you need to add:

```
import javax.swing.event.MenuEvent;
```

Or

```
import javax.swing.event.*;
```

Or use

```
javax.swing.event.MenuEvent instead of MenuEvent
```


Class importation (2)

- What if you have a name conflict?

E.g: java.util.Date and java.sql.Date

```
import java.util.*;
import java.sql.*;
Date today = new Date( );
//ERROR:java.util.Date or java.sql.Date?
```

- if you only need to refer to one of them, import that class explicitly

```
import java.util.*;
import java.sql.*;
import java.util.Date;
Date today = new Date( ); // java.util.Date
```

- if you need to refer to both of them, you have to use the full package name before the class name

```
import java.util.*;
import java.sql.*;

java.sql.Date today = new java.sql.Date( );
java.util.Date nextDay = new java.util.Date( );
```

or

```
import java.util.*;
import java.sql.*;
Import java.util.Date;

java.sql.Date today = new java.sql.Date( );
Date nextDay = new Date( );
```

Import of Static Members

Consider following code:

```
import java.lang.Math;

public class ImportTest {
    double x = sqrt(1.44);
}
```

Compile:

```
%> javac ImportTest.java
ImportTest.java:3: cannot find symbol
symbol   : method sqrt(double)
location: class importTest
double x = sqrt(1.44);
           ^
1 error
```



For the static members, you need to refer them as **className.memberName**

Static importation

- In J2SE 5.0, importation can also be applied on static fields and methods, not just classes. You can directly refer to them after the static importation.

- E.g. import all static fields and methods of the Math class

```
import static java.lang.Math.*;  
double x = PI;
```

- E.g. import a specific field or method

```
import static java.lang.Math.sqrt;  
public class ImportTest {  
    double x = sqrt(1.44);  
}
```

Now compiles without errors;

- Any version before J2SE 5.0 does NOT have this feature!
- **Try** to avoid using this. Makes code harder to read

The default package

- Compilation units (files) that do not declare a package are put into a default, unnamed, package.
- Classes in the default package:
 - Cannot be imported
 - Cannot be used by classes in other packages
- Many editors discourage the use of the default package.

Encapsulation of classes into a package

- Add a class into a package — two steps:
 1. put the name of the package at the top of your source file
 - Use **package** statement to define the package of a class
 - **Must** be the first statement of a file

```
package com.hostname.corejava;  
public class Employee {  
    . . .  
}
```

2. Store .java and .class files in a directory tree that mimics package structure

→ stored in the file "Employee.java" which is stored under "somePath/com/hostname/corejava/"

How to store files and packages?

- Class files must be stored in a directory structure that mirrors package hierarchy
 - class \leftrightarrow file
 - package \leftrightarrow directory (folder)
 - sub-package \leftrightarrow sub-directory into directory of parent package (folder)
- Location of class a.b.c.D?
 - **a/b/c/D.class**
 - Somewhere in the file system!
 - Usually, it is relative to the root of your project
- Compiled classes can be stored in different locations in the file systems
- How does the Java finds them to compile and run?

Jar Files

- **JAR: Java ARchive.** A group of Java classes and supporting files combined into a single file compressed with ZIP format, and given .JAR extension.
- Advantages of JAR files:
 - compressed; quicker download
 - just one file; less mess
 - can be executable

JAR Archive - Creation

- From the command line:

```
jar -cvf filename.jar files
```

- Example:

- Creates a jar file containing all class files in current directory

```
jar -cvf MyProgram.jar *.class
```

- Creates a jar archive containing the package stored in a given directory

```
jar -cvf MyProgram.jar dirName
```

- The jar file contains all files in dirName and preserves the directory structure

- See the content of a jar file

```
jar -tf MyProgram.jar
```

- Some Linux have a menu option to create jars

- **Do not use it. The structure is wrong!**

How does the virtual machine locate classes?

- How to tell the java virtual machine where to find the .class files?

Answer: set the ***class path***

- Class path is the collection of all directories and archive files that are starting points for locating classes.
- Can include:
 - the current "working directory" from which you ran javac / java
 - other folders
 - JAR and ZIP archives
 - Archive must store class files within the appropriate directory structure
 - URLs
 - ...

Setting the class path

- This is needed both for running and compiling in Java!
- Tedious way:
 - set the class path with the `-classpath` option for the `javac/java` programs
 - In command line
 - `$> java -classpath /home/jcp/project animal.Animal`
 - There is a lazier way
 - `-cp` instead of `-classpath`
 - `$> java -cp /home/jcp/project animal.Animal`
- Set classpath in environment variable `CLASSPATH`
 - `$> export CLASSPATH="/home/jcp/project"`
 - `$> java animal.Animal`

Setting the class path – 2

- Set the CLASSPATH environment variable in a permanent way
 - UNIX/Linux
 - If you use the C shell, add a line such as the following to the .cshrc file in your home directory

```
setenv CLASSPATH /home/user/classdir:.
```
 - If you use bash, add a line such as the following to the .bashrc or .bash_profile file in your home directory

```
export CLASSPATH=$CLASSPATH:./home/user/classdir
```
 - after you save the modified files, run the command

```
source .bashrc(or .cshrc or .bash_profile)
```
 - Windows
 - Open the control panel, then open the **System** icon and select the **Environment** tab. Add a new environment variable named CLASSPATH and specify its value, or edit the variable if it exists already.

Locating files - Example

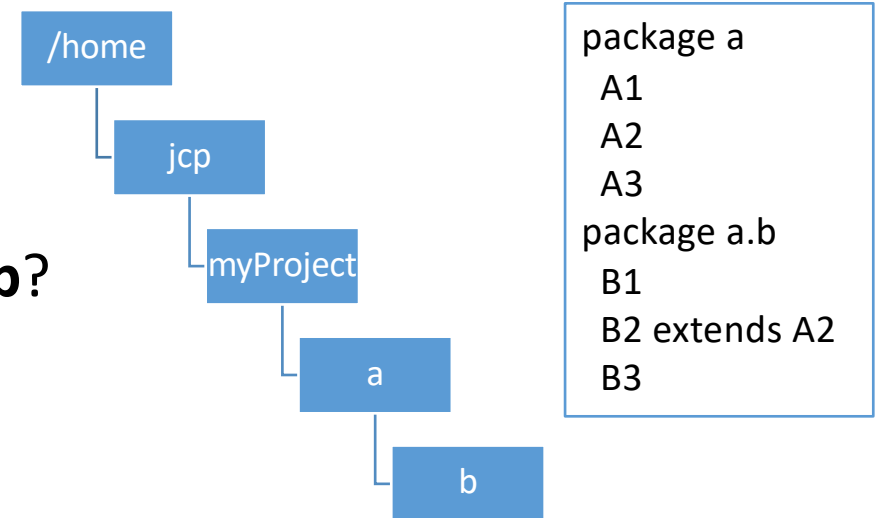
- First suppose the following is the current classpath:
`/home/user/classdir../home/user/archives/archive.jar`
- then suppose the interpreter is searching for the class file of the `pt.tecnico.po.core.Employee` class.
- It will search whether the following files exist in the following order:
 1. `pt/tecnico/po/core/Employee.class` inside archives in `jre/lib` (default)
 2. `pt/tecnico/po/core/Employee.class` inside archives in `jre/lib/ext` (default)
 3. `/home/user/classdir/pt/tecnico/po/core/Employee.class` (first directory in classpath)
 4. `./pt/tecnico/po/core/Employee.class` (second directory in classpath)
 5. `pt/tecnico/po/core/Employee.class` inside `/home/user/archives/archive.jar` (third directory in classpath)
- if any of them is found, then the interpreter stops searching process
- If none is found: **ClassNotFoundException**

Organizational Package Naming Conventions

- Package names should be all lowercase characters whenever possible.
- How to guarantee unique name space for a company?
 - Package names should be made as unique as possible to prevent name clashes
- Frequently a package name begins with the top level domain name of the organization and then the organization's domain and then any subdomains listed in reverse order.
 - Tradition of package name: reverse of the company's Internet domain name
e.g. hostname.com -> com.hostname
- The organization can then choose a specific name for their package.

Compilation Examples

- How to compile all classes in package **a**?
 - In directory `/home/jcp/myProject` do
 - `$> javac a/*.java`
- How to compile all classes in packages **a** and **b**?
 - In directory `/home/jcp/myProject` do
 - `$> javac a/*.java a/b/*.java`
- How to compile all classes in packages **b**?
 - In directory `/home/jcp/myProject` do
 - `$> javac a/b/*.java`
 - In directory `/home/jcp` do
 - `$> javac myProject/a/b/*.java`
 - What happens?
 - Cannot find class `a.A2`
 - `$> javac -cp myProject myProject/a/b/*.java`



Generic compilation command - Unix

- In root project directory
 - `javac -cp <libs> `find rootPackageDir -name *.java``

Common Mistakes

- In directory `/home/jcp/myProject` do
 - `$> java A1.class`
 - Exception in thread "main" `java.lang.NoClassDefFoundError: A1.class`
 - `$> java A1`
 - Exception in thread "main" `java.lang.NoClassDefFoundError: A1`
 - Correct way
 - `$> java a.A1`
- In directory `/home/jcp/` do
 - `$> java a.A1`
 - Exception in thread "main" `java.lang.NoClassDefFoundError: a/A1`
 - Correct way
 - `$> java -cp myProject a.A1`
 - `$> java -cp /home/jcp/myProject a.A1`

```
package a
A1
A2
A3
Package a.b
B1
B2 extends A2
B3
```


Implementation Example

- Um habitat permite guardar aves. Cada habitat tem um número máximo de aves que pode guardar. A adição de uma nova ave deve indicar se teve sucesso ou não. Deve ser possível saber o número de aves do habitat. O habitat pode dizer as todas as aves para voarem. Esta classe deve pertencer ao package habitat.
- Uma ave tem um nome e pode voar. Quando voa escreve a mensagem “Estou a voar”. É possível saber o nome de uma ave. Uma ave está triste se o habitat tiver menos do que 3 aves.
- Um pinguim é uma ave que não sabe voar mas que sabe mergulhar. Quando voa escreve a mensagem “Não sei voar”. Um pinguim quando mergulha fica molhado e só volta a ficar seco quando se secar.
- Pinguim e Ave pertencem ao package animal.