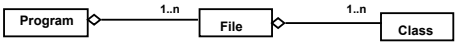
Organization of a java program

SOftEng

Program, files and classes

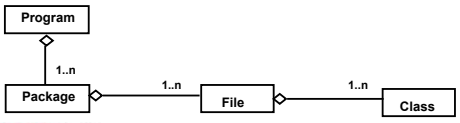
- A program is made of many classes
- A class is in one file
- A file usually contains one class
  - Can also contain more than one, but only one public
- Name of file must be == name of public class



SOftEng

Packages

- Packages add one more level
- Package is implemented via a directory
  - name of directory == name of package



SOftEng

Ex

- Class Helloworld, classe Foo on two files

File HelloWorld.java

```
public class HelloWorld {
        public static void main(String args[]){
                System.out.println("Hello world!");
                Foo b = new Foo();
                b.print();      }}
```

File Foo.java

```
public class Foo {
        public Foo(){}
        public void print(){
           System.out.println("Hello Foo;")
        }
}
```

Ex

- Class Helloworld, class Foo on one file
  - Foo visible only in the package where it is defined

File HelloWorld.java

```
public class HelloWorld {
        public static void main(String args[]){
                System.out.println("Hello world!");
                Foo b = new Foo();
                b.print();      }}


class Foo {
        public Foo(){}
        public void print(){
           System.out.println("Hello Foo;")
        }
}
```

Organization

- Files and folders are a way to organize program
  - And
- Impact visibility

SOftEng

Interpreter vs compiler

SoftEng

Interpreter vs. compiler

- Java: interpreted
- C: compiled

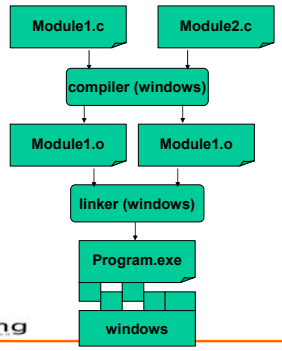
SoftEng

Compiling, running a C program

- Steps
  - Compile (several files)
    - Syntactic check (on each file separately)
    - Translate in machine code (.o)
  - Link
    - Put together .o files (addresses are rearranged)
    - Produces one executable file
  - Load and run
    - load executable (one file)
    - Executable takes control of CPU

SoftEng

C program



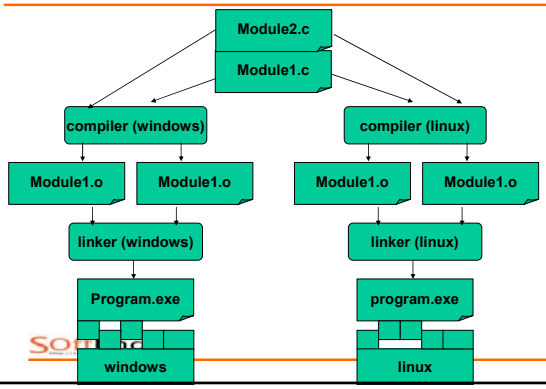
SoftEng

Compilation

- The resulting program depends on the target environment
  - Exe file runs on Windows, not on Linux
- Portability is possible, but requires re-doing all steps
- Static link

SoftEng

C program



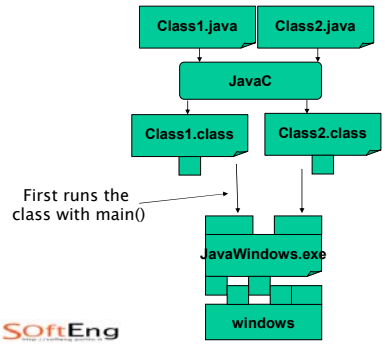
SoftEng

Compiling, running a Java program

- Steps
  - Compile (several files)
    - Syntactic check (on each file separately)
    - Translate in byte code (.class)
  - Run
    - The Java interpreter (JRE) takes control of CPU
    - Only one file is loaded at a time
      - Others are loaded when needed → dynamic loading and linking

SOftEng

Java program



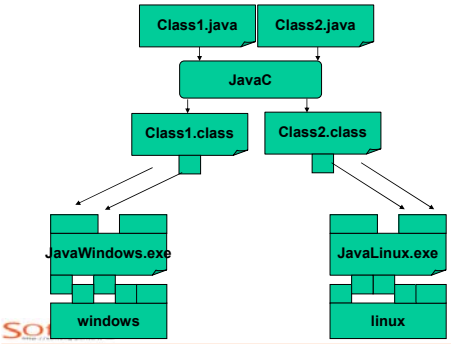
SOftEng

Interpretation

- The interpreter program depends on the target environment
  - JRE runs on Windows, not on Linux
- The .class files run everywhere
- Portability is obtained

SOftEng

Java program



SOftEng

Java

File HelloWorld.java

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

```
> javac HelloWorld.java
> java HelloWorld
HelloWorld!
>
```

compile

execute

SOftEng

Rules for compilation (Java)

- Compilation (syntactic check + production of byte code) is done separately (file by file, or class by class)
- If a class is requested, the compilers looks for it in the same directory

File HelloWorld.java

```
public class HelloWorld {
        public static void main(String args[]){
                System.out.println("Hello world!");
                Foo b = new Foo();
                b.print();      }}
```

SOftEng

- If it is not in the same directory it looks
  - In dir specified by
    - java –classpath <dir> <class.java>
  - In paths specified by development environment (Eclipse)
  - In system libraries
    - ..jre\lib\rt.jar ..jre\lib\i18n.jar

SOftEng

Rules for execution

- Interpreter starts from the class indicated by the programmer, loads it, looks for main() function, starts executing from main

```
> java HelloWorld2
```

Interpreter looks for file HelloWorld2.class, looks for main()

SOftEng

Dynamic loading/linking

- When another class is referred, interpreter looks for file with same name, loads it, executes requested methods

```
> java HelloWorld2
```

Interpreter looks for file e HelloWorld2.class
Loads it, runs main()

```
public class HelloWorld2 {
        public static void main(String args[]){
            System.out.println("Hello world!");
            Foo b = new Foo();
            b.print();      }}
```

Prints Hello World!

Interpreter looks for file Foo.class, loads it, runs b

SOftEng

- Rules for finding files are same as for javac
  - Check directory of project
  - Check class path or paths defined by programmer in Eclipse

SOftEng