Escopo e Pacotes

Three Categories of Data

- **Instance data is the internal representation of a specific object. It records the object's state.**
- **Class data is accessible to all objects of a class.**
- **Local data is specific to a given call of a method.**

Instance Data

Instance data is the internal representation of a specific object.

```
public class Name
{
    // Instance variables
    String first;
    String middle;
    String last;
    . . .
}
```

Class Data

- **Class data is accessible to all objects of a class.**
- **Fields declared as static belong to the class rather than to a specific instance.**

```
public class Name
{
  // Class constant
  static final String PUNCT = ", ";
          . . .
}
```

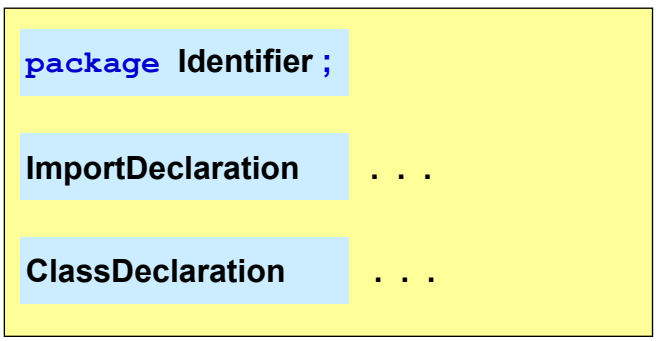
Local Data

- **Local data is specific to a given call of a method.**
- **The JVM allocates space for this data when the method is called and deallocates it when the method returns.**

```
public int compareTo(Name otherName)
{
  int result;     // Local variable
  . . .
  return  result;
}
```

Sintaxe de *Package*

Compilation Unit



Pacote: unidade de *biblioteca*

- Gerenciando o “espaço de nomes”
  - Class members are already hidden inside class
  - Class names could clash
  - Need completely unique name
- Pacotes
  - organize classes into libraries
  - structure name space for classes
  - restrict visibility
  - may be nested

Creating a Library of Classes

```
package mypackage;
public class Class1{ ... }
package mypackage.mysubpackage1;
public class Class5{ ... }
```

- **public** class is under the umbrella **mypackage**
- Client programmer must import the package

```
import mypackage.*;
import java.util.Vector;
```

Compilation Units

- Compilation units (**.java** files)
  - Name of **.java** file == name of single **public** class
  - Other non-**public** classes are not visible
  - Each class in file gets its own **.class** file
  - Program is a bunch of **.class** files

Localização dos pacotes

- Creating unique package names
  - Location on disk encoded into package name
  - Convention: first part of package name is Internet
  - domain name of class creator (reverse)
    sts.tu-harburg.de ➔ de.tu-harburg.sts.mypackage
- Java interpreter
  - uses CLASSPATH environment variable as starting
- point for search
  - looks for package x.y.z in a folder on the path x/y/z
  - CLASSPATH takes care of first part:

```
CLASSPATH=.;D:\JAVA\LIB;C:\DOC\JavaClasses
```

Package Do's and Don't's

- **A compilation unit can have only one public class**
- **Many compilation units can be in a package**
- **"No modifier" specifies package access**
- **Any field or method with package access can be accessed by any member of the package**

Package Example

- **package** `addressBook`
- **Members:**
  - **class** `Address`
  - **class** `Entry`
- **Imported by**
  - **class** `AddressDr`
- **All of the variables have package access**

“Friendly”

- Default access, has no keyword
  - public to other members of the same package,
  - private to anyone outside the package.
- Easy interaction for related classes (that you place in the same package)
- Also referred to as “package access”

public: Interface Access

```
package food.dessert;

public class Cookie {
   public Cookie() {
     System.out.println("Cookie constructor");
   }
   void decorate() { System.out.println("Cookie decorated"); }
}

// Em outro arquivo
import food.dessert.*;
public class Dinner {
   public static void main(String args[]) {
      Cookie c = new Cookie();
      //! c.decorate(); // não tenho acesso
                        //não foi declarada pública
                        // e não é do mesmo pacote
   }
}
```

private: não pode acessar

```
class Resource {
   private static count = 5;
   private Resource() {}
   static Resource makeAResource() {
      if (count > 0) {
         count --; return new Resource();}
      else return null;
   }
}

public class ResourceUser {
   public static void main(String args[]) {
   //! Resource r = new Resource();
   Resource r = Resource.makeAResource(); }
}
```

Class Access

- Classes as a whole can be **public** or "friendly"
- Only one **public** class per file, usable outside the library
- All other classes "friendly," only usable within the library