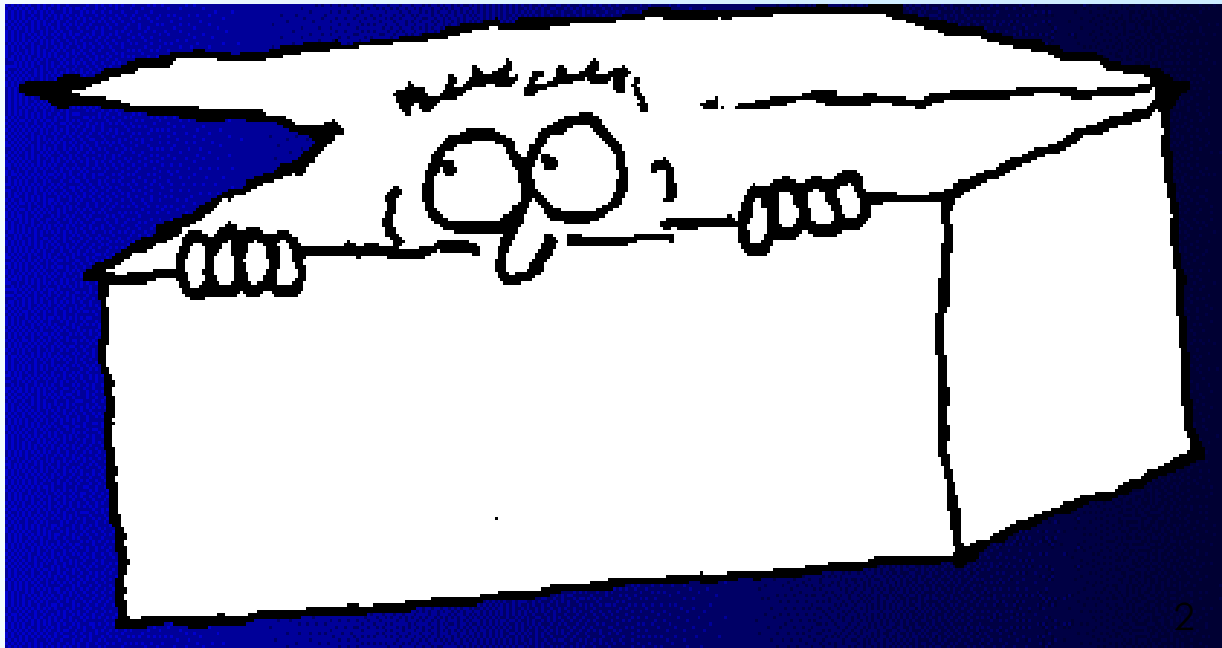
ACCESS CONTROL

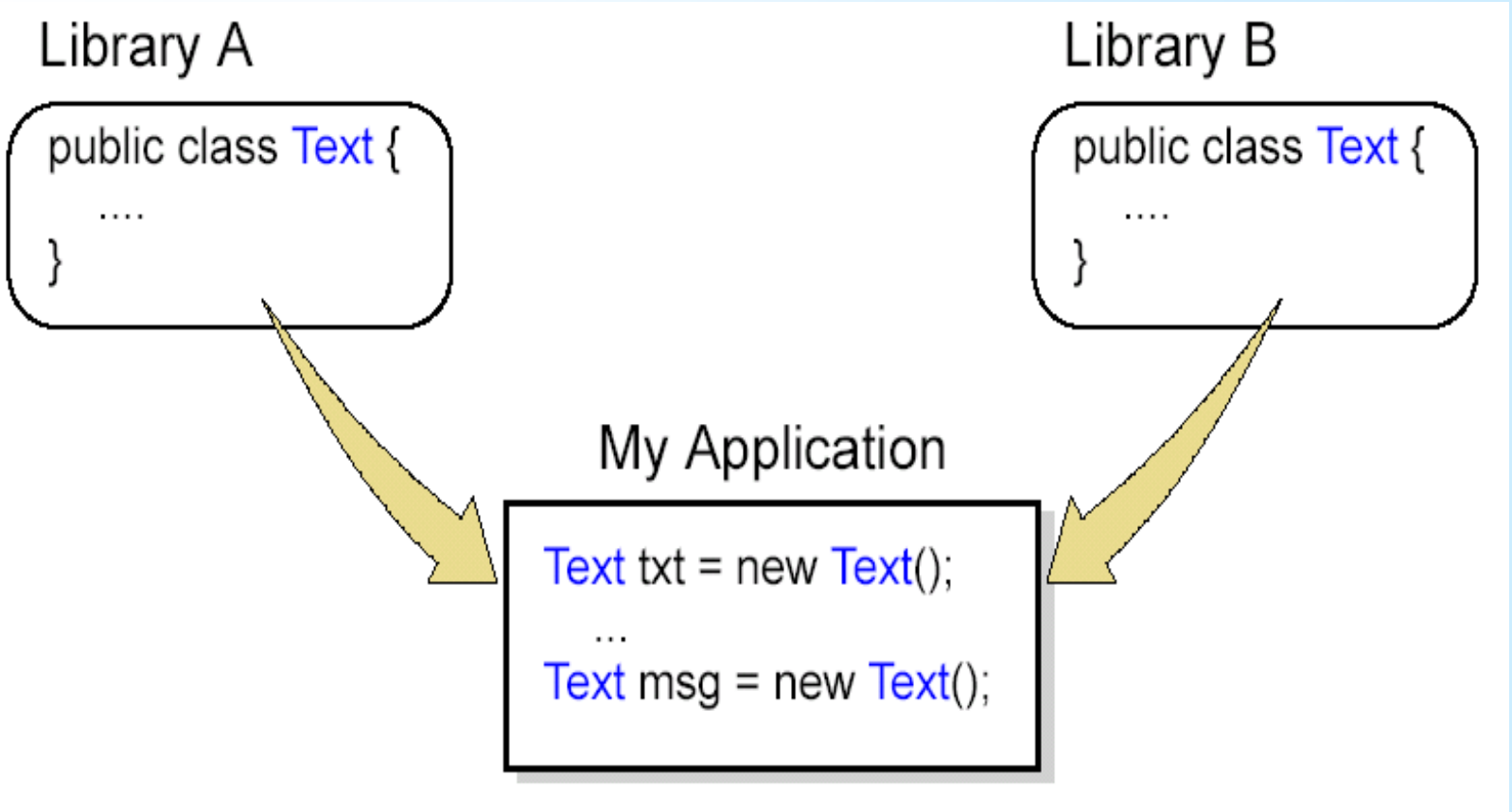
Qiuyan Huo
Software Engineering Institute
qyhuo@mail.xidian.edu.cn

Access Control

- Access control (or hiding the implementation) is about “not getting it right the first time”.



The Naming Problem



`package`

What is Package

- A **`package`** is a collection of functionally related classes and interfaces providing access protection and namespace management.

package: the library unit

```
import java.util.*;
import java.util.ArrayList;
```

- Managing "name spaces"
  - Class members are already hidden inside class
  - Class names could clash *(类名冲突)*
  - Need completely unique name even over the Internet *(名称唯一)*
- 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 (**no .obj | .lib**)

Creating a Library of Classes

`package mypackage;`

- public class is under the umbrella **`mypackage`**
- Client programmer must **`import mypackage.*;`**

- Creating unique package names
  - Location on disk encoded into package name
  - Convention: first part of package name is Internet domain name of class creator (reversed)
  - Java interpreter uses CLASSPATH environment variable as starting point for search
  - **`bruceeckel.com`** (Note change to lowercase '**com**')

```
package com.bruceeckel.util;
```

Legalizing Package Names

Domain Name	Package Name Prefix
`hyphenated-name.example.org`	`org.example.hyphenated_name`
`example.int`	`int_.example`
`123name.example.com`	`com.example._123name`

A Simple Library

```
package com.bruceeckel.util;
public class Vector {
  public Vector() {
      System.out.println(
          "com.bruceeckel.util.Vector");
  }
}
///////////////////////////////////
package com.bruceeckel.util; //Separate file
public class List {
  public List() {
      System.out.println(
          "com.bruceeckel.util.List");
  }
}
```

Library Location

```
C:\DOC\JavaT\com\bruceeckel\util
```

- CLASSPATH takes care of first part:

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

- Programs can be in any directory

```
import com.bruceeckel.util.*;
public class LibTest {
      public static void main(String args[]) {
            Vector v = new Vector();
            List l = new List();
      }
}
```

Collisions?

- Compiler starts search at CLASSPATH

Beyond Basic Arithmetic

- **`Math`** in **`java.lang`** package
  - methods (>40) are all static, i.e,
    - **`Math.cos(angle);`**
    - **`Math.random(); // [0.0 ~ 1.0)`**
  - two constants
    - **`Math.E`** : the base of natural logarithms
    - **`Math.PI`**: the ratio of the circumference of a circle to its diameter

```
class Math {

    public static final double PI = 3.141592653589793;
    public static double cos(double a){
        //...
    }
```

```
//...
double r = Math.cos(Math.PI * theta);
```

Static Import

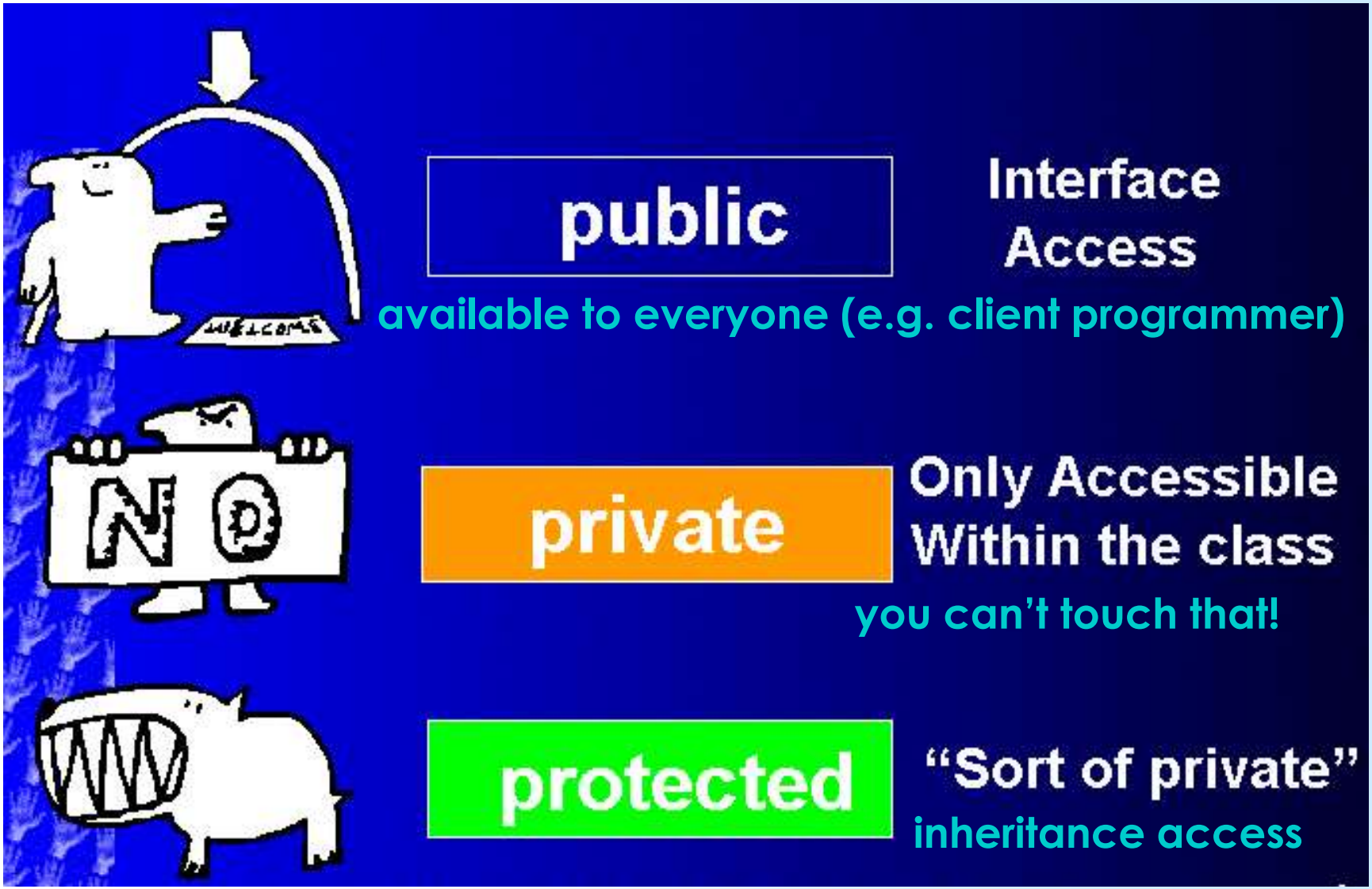
- import the constants and static methods
  - need frequent access to static final fields (constants) and static methods from one or two classes.
  - do not need to prefix the name of their class

```
import static java.lang.Math.PI;

import static java.lang.Math.*;
```

```
double r = cos(PI * theta);
```

Java Access Specifiers

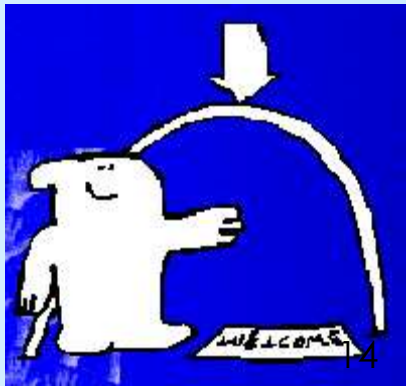


"Friendly"

- ***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 access.dessert;
public class Cookie {
  public Cookie() {
   System.out.println("Cookie constructor");
  }
  void bite() { System.out.println("bite"); }
} ///:~
//separate file in separate package
import access.dessert.*;
public class Dinner {
  public Dinner() {
      System.out.println("Dinner constructor");
  }
  public static void main(String[] args){
    Cookie x = new Cookie();
    //! x.bite(); // Can't access
  }
} ///:~
```



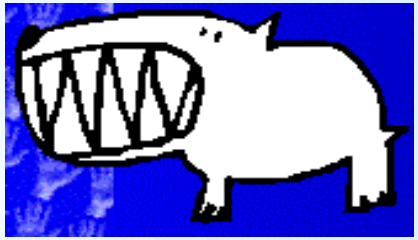
private: Can't Touch That!

```
class Sundae {
  private Sundae() {}
  static Sundae makeASundae() {
    return new Sundae();
  }
}

public class IceCream {
  public static void main(String[] args) {
    Sundae x = new Sundae();
    Sundae x = Sundae.makeASundae();
  }
} ///:~
```

Sundae() has private access in Sundae

`protected`



- deals with inheritance
- Generally: when you need it, you'll know
- Covered later

Class Access

```
public class Widget { //…package access;

import access.Widget;
or
import access.*;
```

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

Note that a class **cannot** be **private** (that would make it accessible to no one but the class) or **`protected`**.

Top 10 Mistakes Java Developers Make: #6

- Very often developers use public for class field.
- It is easy to get the field value by directly referencing, but this is a very bad design.
- The rule of thumb is giving access level for members as low as possible.

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	X
no modifier	Y	Y	X	X
private	Y	X	X	X

Summary of Hiding the Implementation

- Access control tells users what they can & can't use (shows the area of interest)
- Also separates interface & implementation
- Allows the class creator to change the implementation later without disturbing client code
- An important design & implementation flexibility
- Design guideline: always make elements "as private as possible"