Object Oriented Programming

Encapsulation – Inheritance I

Programación II
Facultad de Ingeniería
Universidad Austral

OOP Concepts

- **OOP provides the programmer with a number of important concepts:**
  - Modularity
  - Code Re-Use
  - Encapsulation
  - Inheritance
  - Polymorphism
  - Let's look at these more closely...

Modularity and Code Re-Use

- You've long been taught to break down complex problems into more tractable sub-problems.
- Each class represents a sub-unit of code that (if written well) can be developed, tested and updated independently from the rest of the code.
- Indeed, two classes that achieve the same thing (but perhaps do it in different ways) can be swapped in the code
- Properly developed classes can be used in other programs without modification.

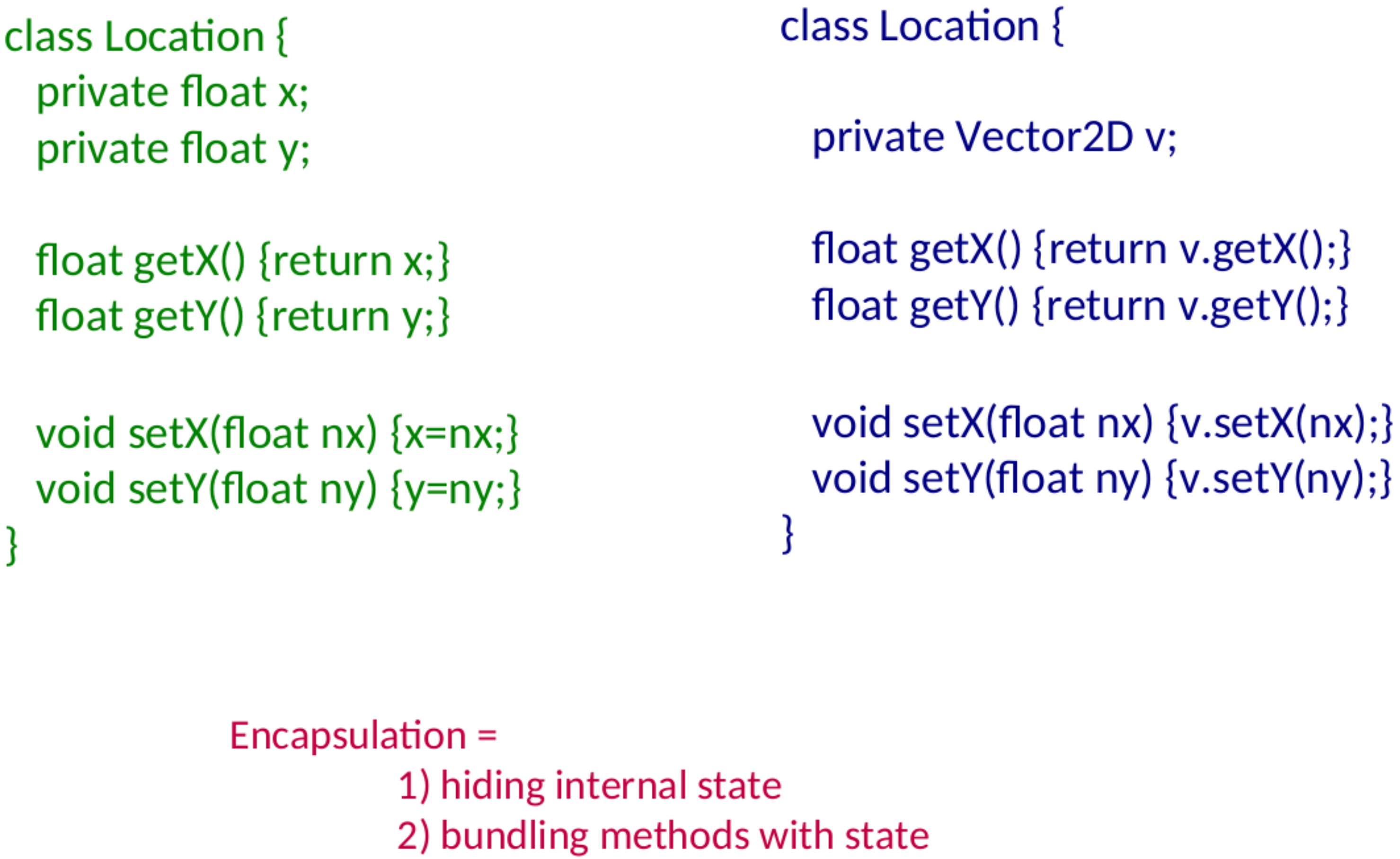
Encapsulation I

```
class Student {
    int age;
};
void main() {
    Student s = new Student();
    s.age = 21;
    Student s2 = new Student();
    s2.age=-1;
    Student s3 = new Student();
    s3.age=10055;
}
```

Encapsulation II

```
class Student {
    private int age;
    boolean setAge(int a) {
        if (a>=0 && a<130) {
            age=a;
            return true;
        }
        return false;
    }
    int getAge() {return age;}
}
void main() {
    Student s = new Student();
    s.setAge(21);
}
```

Encapsulation III



```
class Location {
   private float x;
   private float y;

   float getX() {return x;}
   float getY() {return y;}

   void setX(float nx) {x=nx;}
   void setY(float ny) {y=ny;}
}
```

```
class Location {

   private Vector2D v;

   float getX() {return v.getX();}
   float getY() {return v.getY();}

   void setX(float nx) {v.setX(nx);}
   void setY(float ny) {v.setY(ny);}
}
```

Encapsulation =

1) hiding internal state
2) bundling methods with state

Packages in Java

- **Package in Java is a mechanism to encapsulate a group of classes, sub packages and interfaces. Packages are used for:**
  - Preventing naming conflicts. For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee
  - Making searching/locating and usage of classes, interfaces, enumerations and annotations easier.

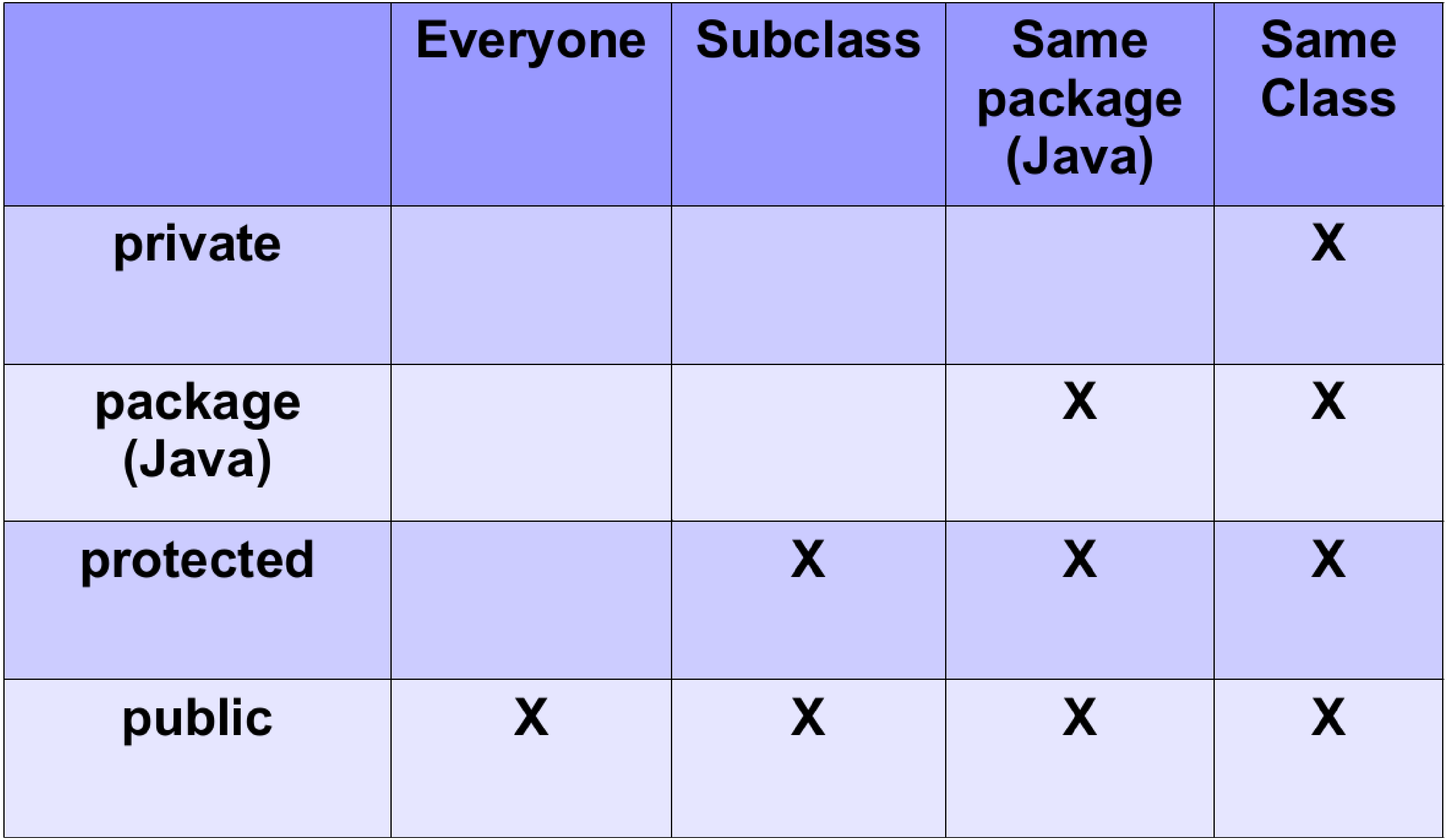
Packages in Java

- **Packages are used for:**
  - Providing controlled access:
    - Protected and default have package level access control.
    - A protected member is accessible by classes in the same package and its subclasses.
    - A default member (without any access specifier) is accessible by classes in the same package only.
  - Packages can be considered as data encapsulation (or data-hiding).

Packages in Java

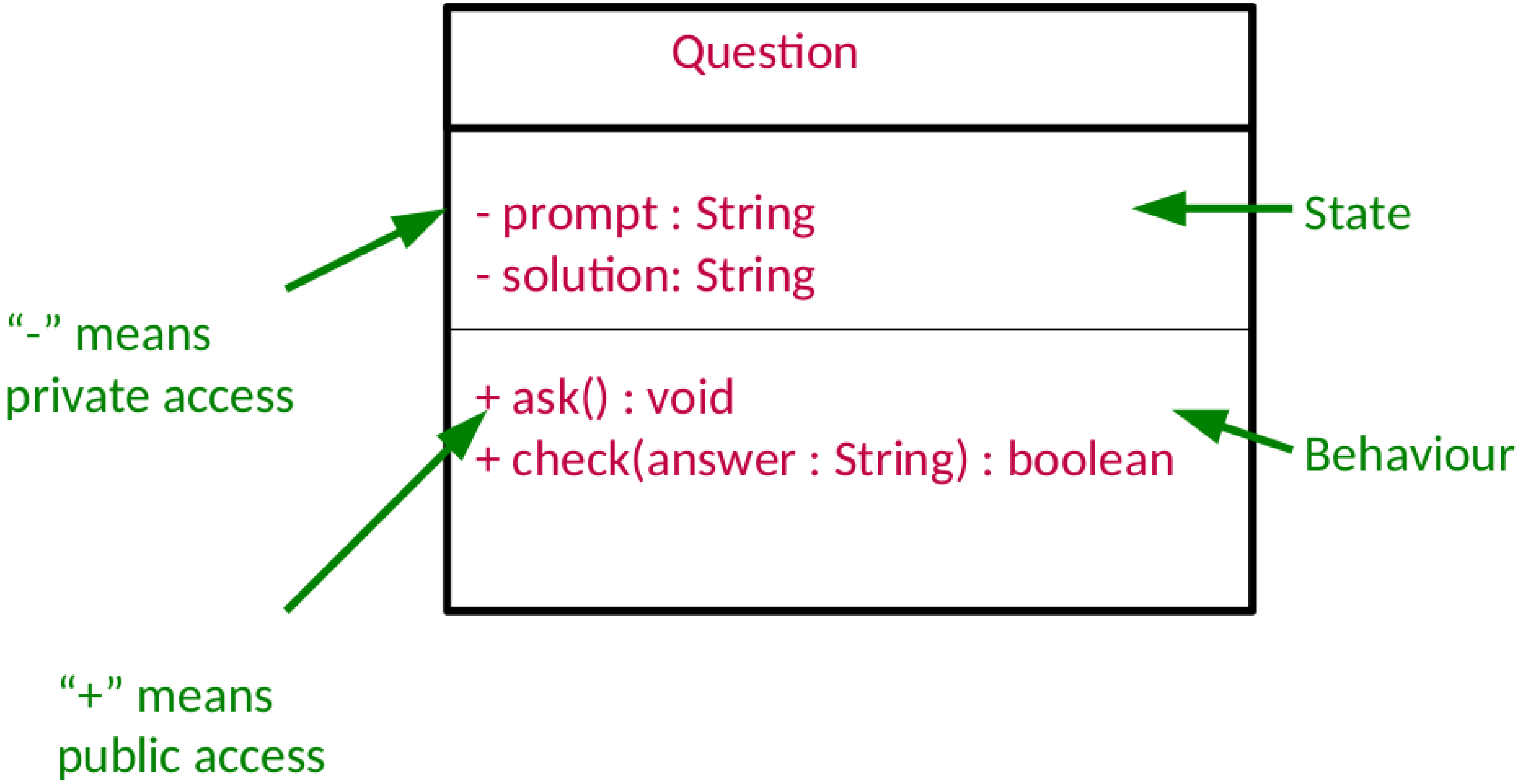
- **All we need to do is put related classes into packages.**
  - After that, we can simply write an import class from existing packages and use it in our program.
  - A package is a container of a group of related classes where some of the classes are accessible are exposed and others are kept for internal purpose.
- **We can reuse existing classes from the packages as many time as we need it in our program.**

Access Modifiers



	Everyone	Subclass	Same package (Java)	Same Class
private				X
package (Java)			X	X
protected		X	X	X
public	X	X	X	X

UML: Representing a Class Graphically



Question

- prompt : String
- solution: String

+ ask() : void
+ check(answer : String) : boolean

State

Behaviour

“-” means private access

“+” means public access

Identifying Classes

- **We want our class to be a grouping of conceptually related state and behaviour**
  - One popular way to group is using grammar
    - Noun → Object
    - Verb → Method
- **“A quiz program that asks questions and checks the answers are correct”**

Inheritance I

```
class Student {
   private int age;
   private String name;
   private int grade;
   …
}

class Lecturer {
   private int age;
   private String name;
   private int salary;
   …
}
```

- There is a lot of duplication here
- Conceptually there is a hierarchy that we're not really representing
- Both Lecturers and Students are people (no, really).
- We can view each as a kind of specialisation of a general person
  - They have all the properties of a person
  - But they also have some extra stuff specific to them

Inheritance II

```
class Person {
  protected int age;
  protected String name;
  ...
}

class Student extends Person {
  private int grade;
  ...
}

class Lecturer extends Person {
  private int salary;
  ...
}
```

- We create a *base class* (Person) and add a new notion: classes can *inherit* properties from it
  - Both state, functionality and type
- We say:
  - Person is the *superclass* of Lecturer and Student
  - Lecturer and Student *subclass* Person

‘extends’ in Java gives you both code- and type-inheritance

Note: Java is a **nominative** type language (rather than a **structurally** typed one)

If you mark a class ‘final’ then it can’t be extended and ‘final’ methods can’t be overridden

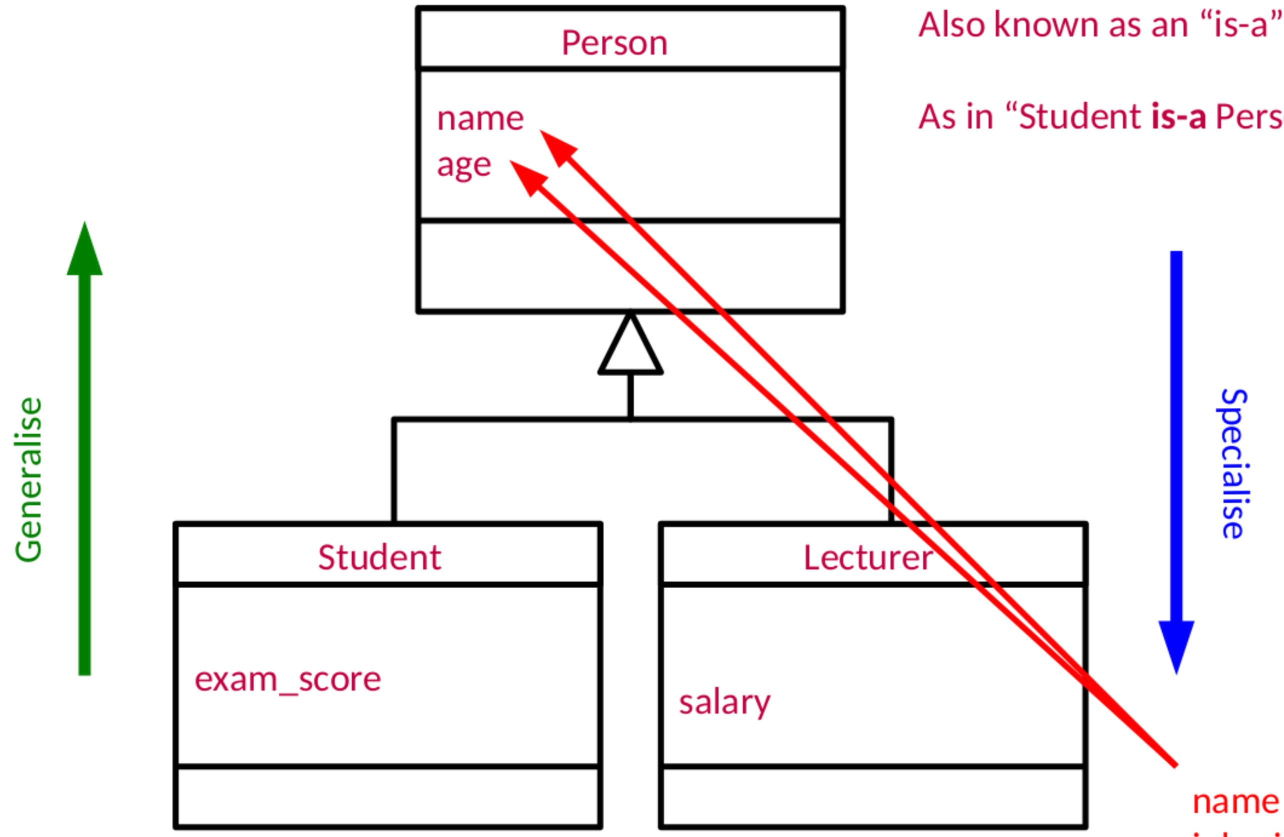
Liskov Substitution Principle

- If S is a subtype of T then objects of type T may be replaced with objects of type S
- Student is a subtype of Person so anywhere I can have a Person I can have a Student instead

Representing Inheritance Graphically

Also known as an “is-a” relation

As in “Student **is-a** Person”



name and age inherited if not private

Questions?