CLASS RELATIONSHIP

ITX 2001, CSX 3002, IT 2371

RELATIONSHIP AMONG CLASSES

1. Inheritance (Superclass-Subclass / Parent-Child / Is-A Relationship)
    - 1.1) Abstract class (Is-An-Extension-of relationship)
    - 1.2) Interface class (Is-A-Kind-of relationship)
2. Association (Binary Relationship)
    - 2.1) Aggregation (Has-A relationship)
    - 2.2) Composition (Is-A-Part of relationship)

1. INHERITANCE (SUPERCLASS-SUBCLASS / PARENT-CHILD / IS-A RELATIONSHIP)

1.1) Abstract class (Is-An-Extension-of relationship)

1.2) Interface class (Is-A-Kind-of relationship)

1.1) ABSTRACT CLASS

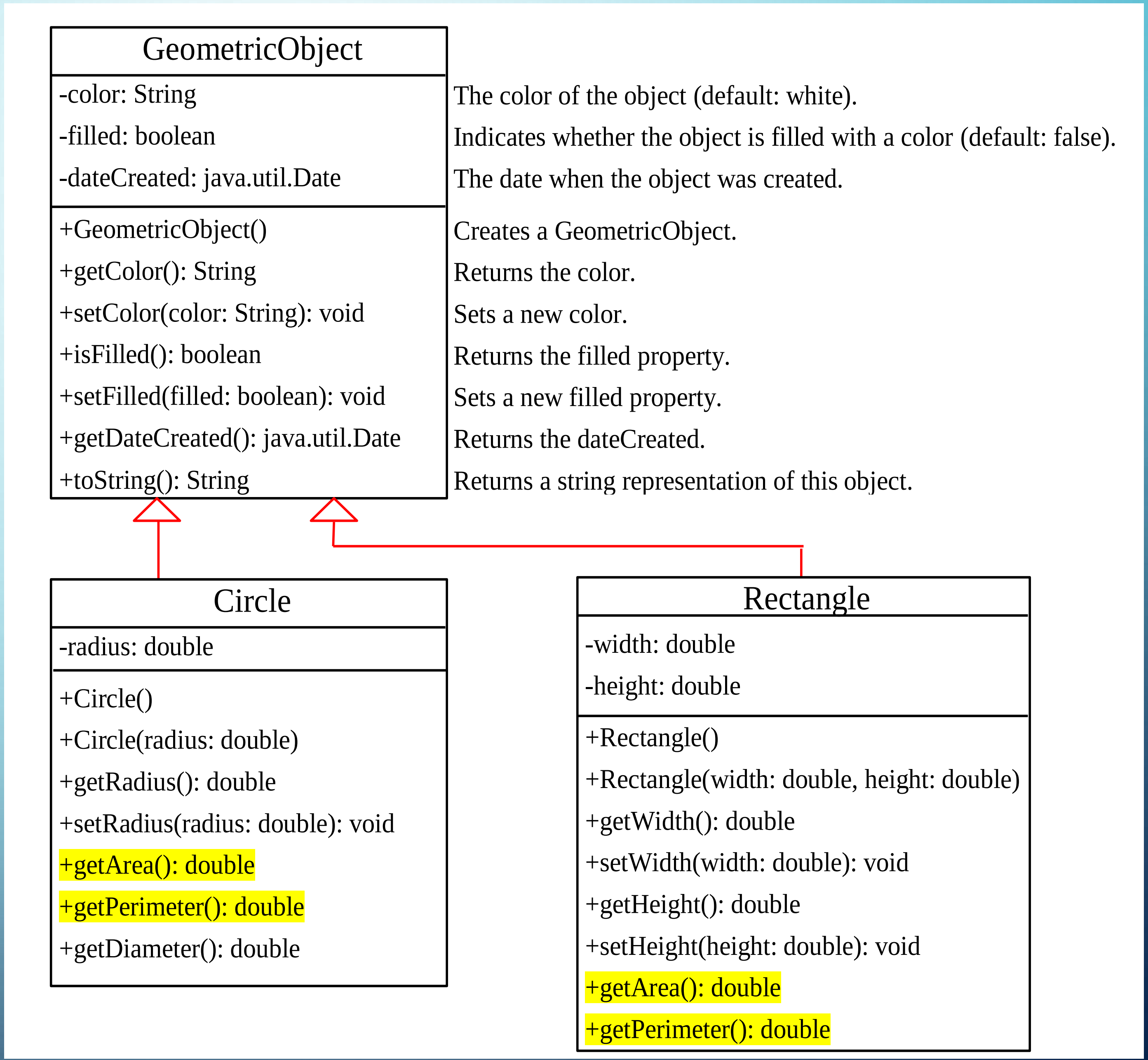
- In an inheritance hierarchy,
  - Root class becomes more general and less specific.
  - Non-leaf or Leaf class are more specific and concrete
- Superclass should contain common features of its subclasses.
- A superclass is so abstract if it cannot have specific instances.

ABSTRACT CLASS PROPERTIES

- It cannot be instantiated.
  - Test invalid code:
    - Geometric objGeometric = new Geometric();
- It must be extended in subclass.
- The abstract method
  - It must be implemented in its subclasses instead of the abstract class itself.

UML NOTATION

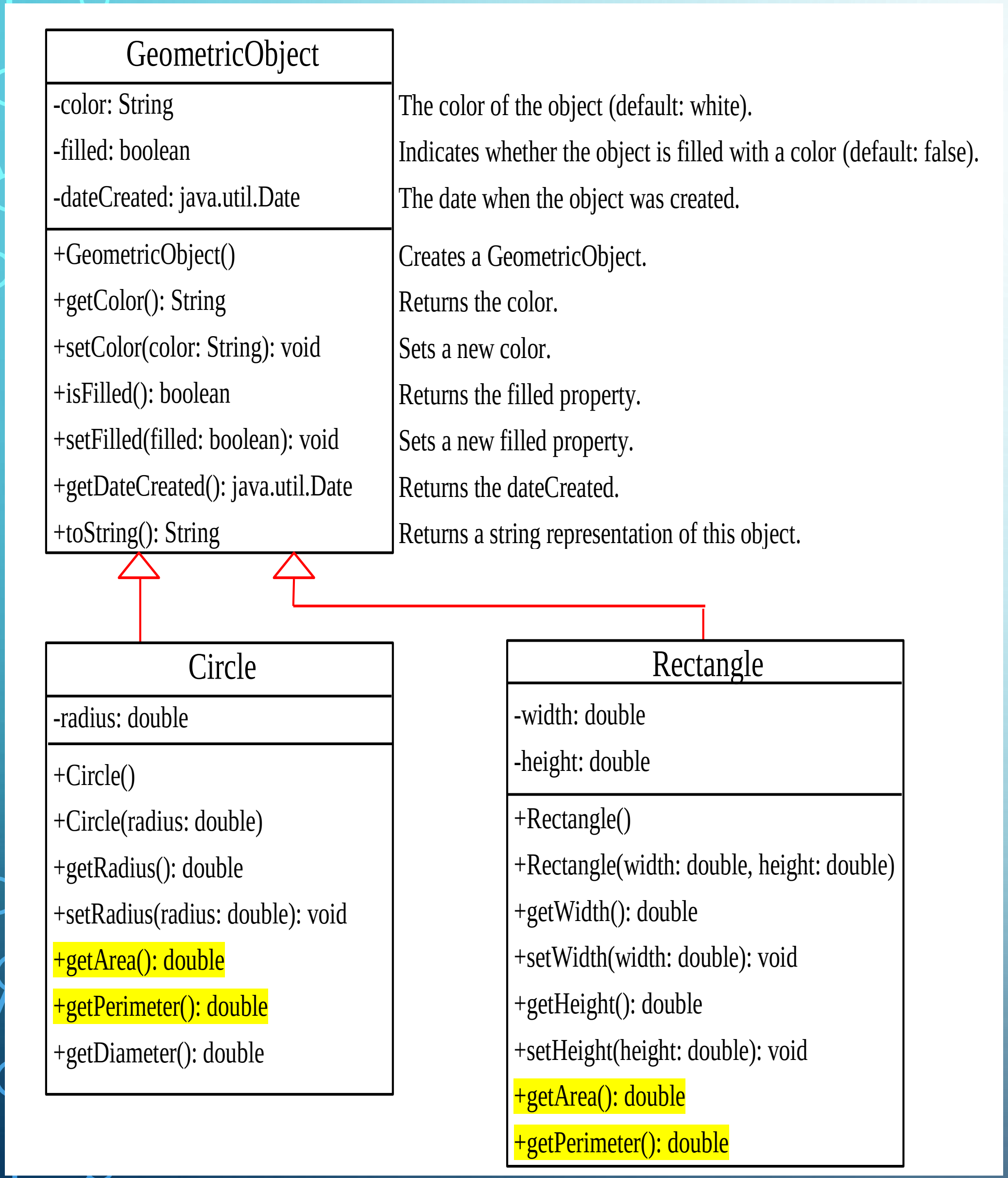
- Italicize name of
  - Abstract class
  - Abstract method

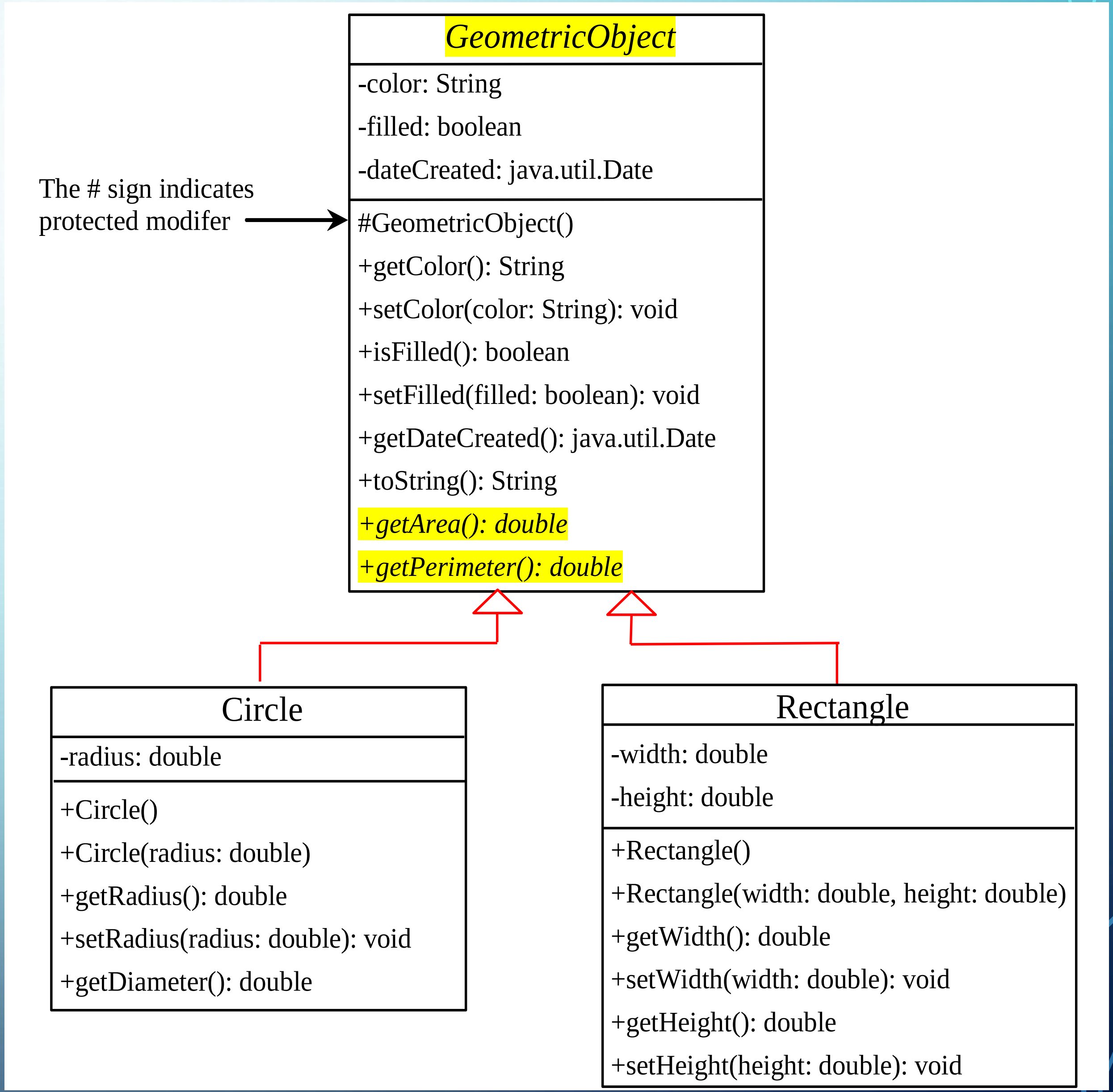


GeometricObject	
-color: String	The color of the object (default: white).
-filled: boolean	Indicates whether the object is filled with a color (default: false).
-dateCreated: java.util.Date	The date when the object was created.
+GeometricObject()	Creates a GeometricObject.
+getColor(): String	Returns the color.
+setColor(color: String): void	Sets a new color.
+isFilled(): boolean	Returns the filled property.
+setFilled(filled: boolean): void	Sets a new filled property.
+getDateCreated(): java.util.Date	Returns the dateCreated.
+toString(): String	Returns a string representation of this object.

Circle
-radius: double
+Circle()
+Circle(radius: double)
+getRadius(): double
+setRadius(radius: double): void
+getArea(): double
+getPerimeter(): double
+getDiameter(): double

Rectangle
-width: double
-height: double
+Rectangle()
+Rectangle(width: double, height: double)
+getWidth(): double
+setWidth(width: double): void
+getHeight(): double
+setHeight(height: double): void
+getArea(): double
+getPerimeter(): double


GeometricObject
-color: String
-filled: boolean
-dateCreated: java.util.Date
+GeometricObject()
+getColor(): String
+setColor(color: String): void
+isFilled(): boolean
+setFilled(filled: boolean): void
+getDateCreated(): java.util.Date
+toString(): String
The color of the object (default: white).
Indicates whether the object is filled with a color (default: false).
The date when the object was created.
Creates a GeometricObject.
Returns the color.
Sets a new color.
Returns the filled property.
Sets a new filled property.
Returns the dateCreated.
Returns a string representation of this object.
Circle
-radius: double
+Circle()
+Circle(radius: double)
+getRadius(): double
+setRadius(radius: double): void
+getArea(): double
+getPerimeter(): double
+getDiameter(): double
Rectangle
-width: double
-height: double
+Rectangle()
+Rectangle(width: double, height: double)
+getWidth(): double
+setWidth(width: double): void
+getHeight(): double
+setHeight(height: double): void
+getArea(): double
+getPerimeter(): double


The # sign indicates protected modifer
GeometricObject
-color: String
-filled: boolean
-dateCreated: java.util.Date
#GeometricObject()
+getColor(): String
+setColor(color: String): void
+isFilled(): boolean
+setFilled(filled: boolean): void
+getDateCreated(): java.util.Date
+toString(): String
+getArea(): double
+getPerimeter(): double
Circle
-radius: double
+Circle()
+Circle(radius: double)
+getRadius(): double
+setRadius(radius: double): void
+getDiameter(): double
Rectangle
-width: double
-height: double
+Rectangle()
+Rectangle(width: double, height: double)
+getWidth(): double
+setWidth(width: double): void
+getHeight(): double
+setHeight(height: double): void

ABSTRACT CLASS AND METHOD CONDITIONS

- An abstract method cannot be in a non-abstract class.
  - A class that contains an abstract method must be an abstract class.
  - An abstract method can contain non-abstract method.
- If a subclass does not implement <u>all</u> abstract methods, it must be declare itself to be the abstract class.
  - Non-abstract class must implement all abstract methods declared in its superclass.

ABSTRACT CLASS CONSTRUCTOR

- An abstract class cannot initiate its instance.
  - However, its constructor can be provided for attributes’ initialization and invoked by its subclass.

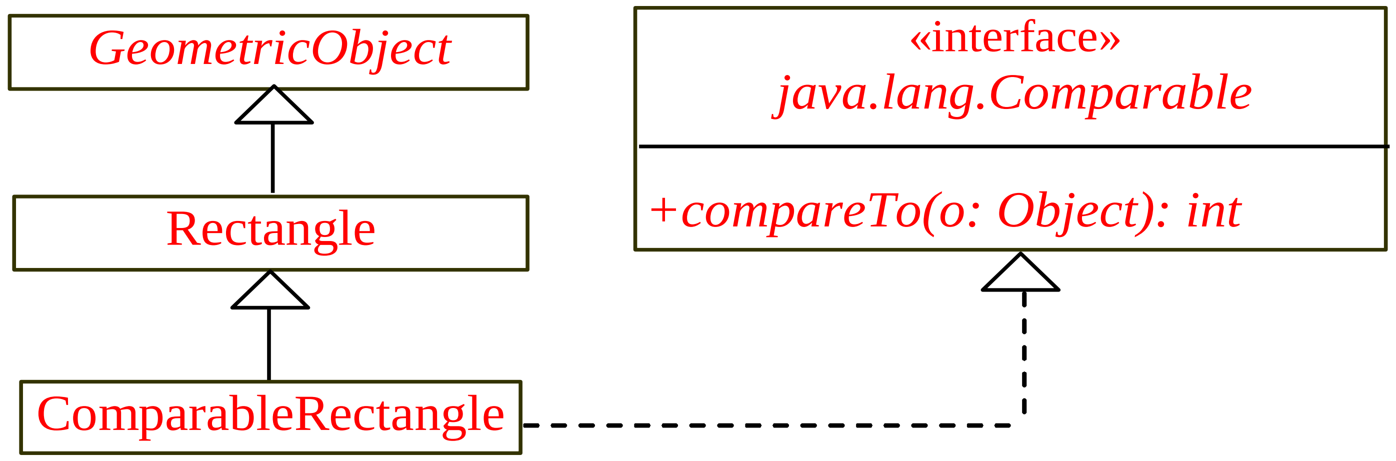
1.2) INTERFACE CLASS

- It is used to specify common behaviors for objects.
- Using appropriate interface class, you can specify those object are comparable, editable and clonal.
- It contains only:
  - Constant,
  - Abstract method
- Keyword “interface” is defined to replace “class”.

UML NOTATION

- Insert keyword: <<interface>> in front of the italicized class name.
- The dash line arrow is used to declare superclass – subclass relationship.

Notation:
The interface name and the method names are italicized. The dashed lines and hollow triangles are used to point to the interface.



Liang, “Introduction to Java Programming” 6th Edition, Pearson Education, 2007

INTERFACE CLASS CONDITIONS

- It is more flexible than abstract class.
    - A subclass can extend only one superclass but implement many interfaces.
- It cannot contain a concrete method.

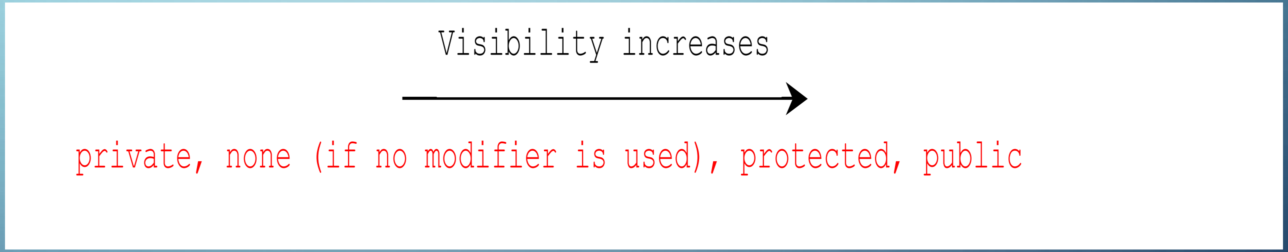
INTERFACE CLASS AND ABSTRACT CLASS

	Variables	**Constructors**	**Methods**
Abstract class	No restrictions	Constructors are invoked by subclasses through constructor chaining. An abstract class cannot be instantiated using the new operator.	No restrictions.
Interface	All variables must be public static final	No constructors. An interface cannot be instantiated using the new operator.	All methods must be public abstract instance methods

Liang, "Introduction to Java Programming" 6th Edition, Pearson Education, 2007

ACCESSIBILITY OF SUPERCLASS-SUBCLASS

- Superclass-subclass apply “protected” modifier to serve the information hiding for both attributes and methods.
- There are possible four authentications:
  1. Private
  2. Default
  3. Protected
  4. Public



Visibility increases

$\xrightarrow{\qquad\qquad\qquad}$

private, none (if no modifier is used), protected, public

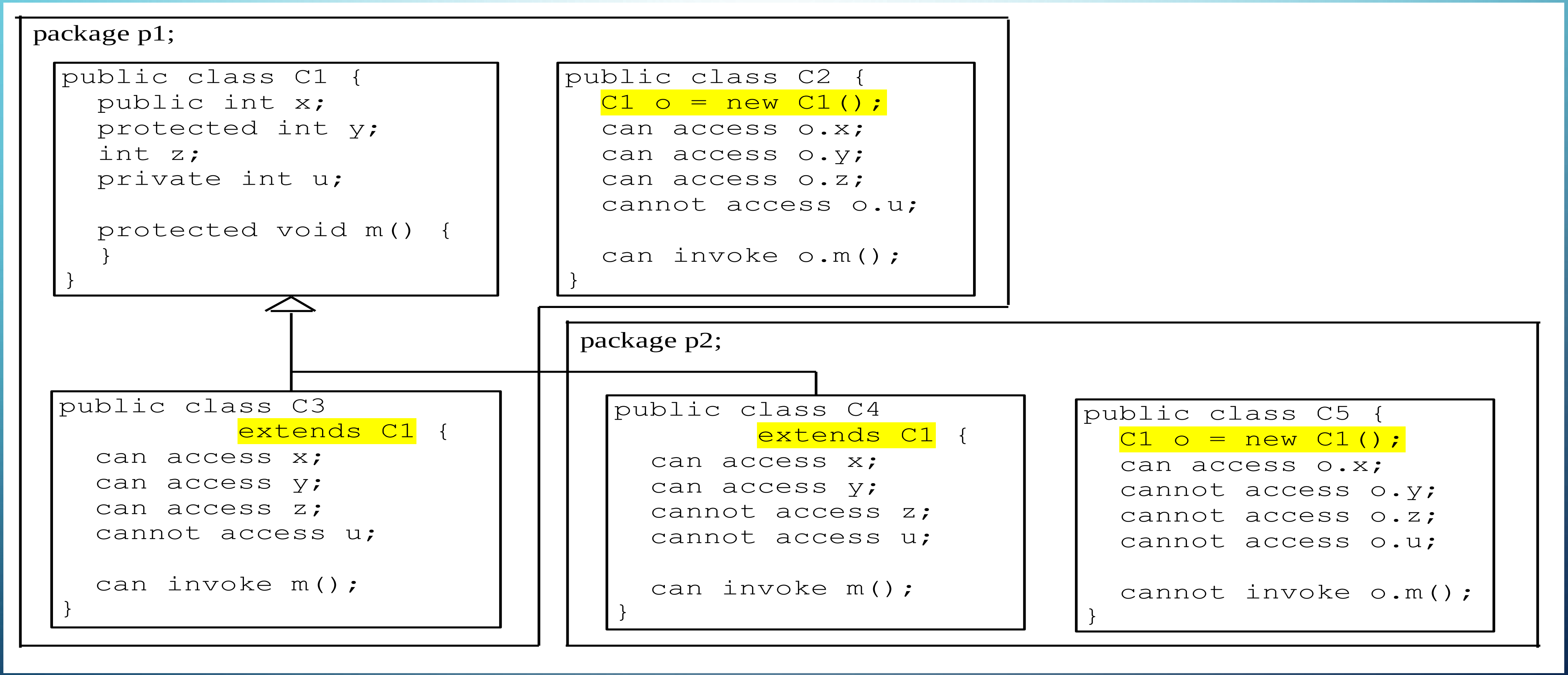
Liang, “Introduction to Java Programming” 6th Edition, Pearson Education, 2007

ACCESSIBILITY SUMMARY

Modifier on members in a class	Accessed from the same class	Accessed from the same package	Accessed from a subclass	Accessed from a different package
public	✓	✓	✓	✓
protected	✓	✓	✓	–
default	✓	✓	–	–
private	✓	–	–	–

Liang, “Introduction to Java Programming” 6th Edition, Pearson Education, 2007

VISIBILITY MODIFIERS



package p1;

```
public class C1 {
  public int x;
  protected int y;
  int z;
  private int u;

  protected void m() {
  }
}
```

```
public class C2 {
  C1 o = new C1();
  can access o.x;
  can access o.y;
  can access o.z;
  cannot access o.u;

  can invoke o.m();
}
```

```
public class C3
          extends C1 {
  can access x;
  can access y;
  can access z;
  cannot access u;

  can invoke m();
}
```

package p2;

```
public class C4
          extends C1 {
  can access x;
  can access y;
  cannot access z;
  cannot access u;

  can invoke m();
}
```

```
public class C5 {
  C1 o = new C1();
  can access o.x;
  cannot access o.y;
  cannot access o.z;
  cannot access o.u;

  cannot invoke o.m();
}
```

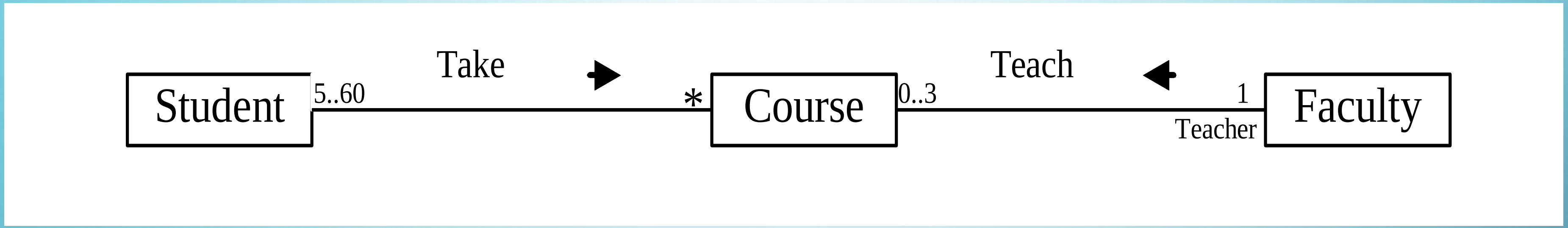
WEAKEN ACCESSIBILITY BY SUBCLASS

- A subclass can override a protected method in its superclass <u>and</u> change its visibility to public.
- In the contrast way, if a superclass method is defined as public, it must be defined as public in the subclass.

2. ASSOCIATION (BINARY RELATIONSHIP)

- It represents a general binary relationship.
- It describes an activity between two classes.
- The class attributes (or fields) will usually move to be a member of the associated class as well.

2. ASSOCIATION



```
public class Student {
  /** Data fields */
  private Course[]
    courseList;

  /** Constructors */
  /** Methods */
}
```

```
public class Course {
  /** Data fields */
  private Student[]
    classList;
  private Faculty faculty

  /** Constructors */
  /** Methods */
}
```

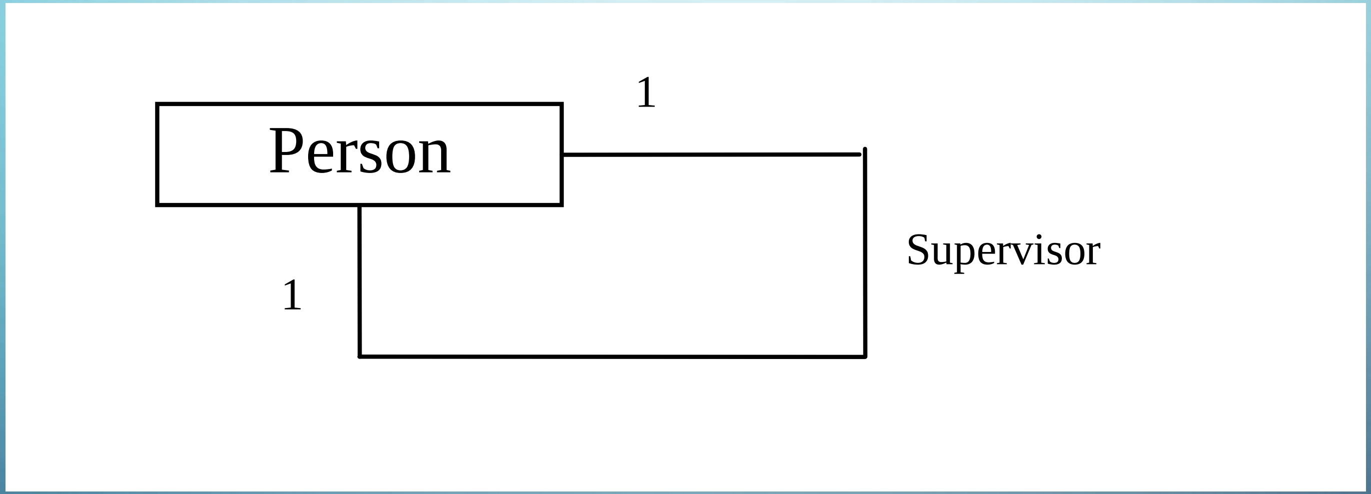
```
public class Faculty {
  /** Data fields */
  private Course[]
    courseList;

  /** Constructors */
  /** Methods */
}
```

Liang, "Introduction to Java Programming" 6th Edition, Pearson Education, 2007

2.1 ASSOCIATION

- It may associate with itself.



Liang, “Introduction to Java Programming” 6th Edition, Pearson Education, 2007

2. ASSOCIATION

- The degree of association can be described in details with two following forms:

2.1) Aggregation (Has-A relationship)

2.2) Composition (Is-a-Part-of relationship)

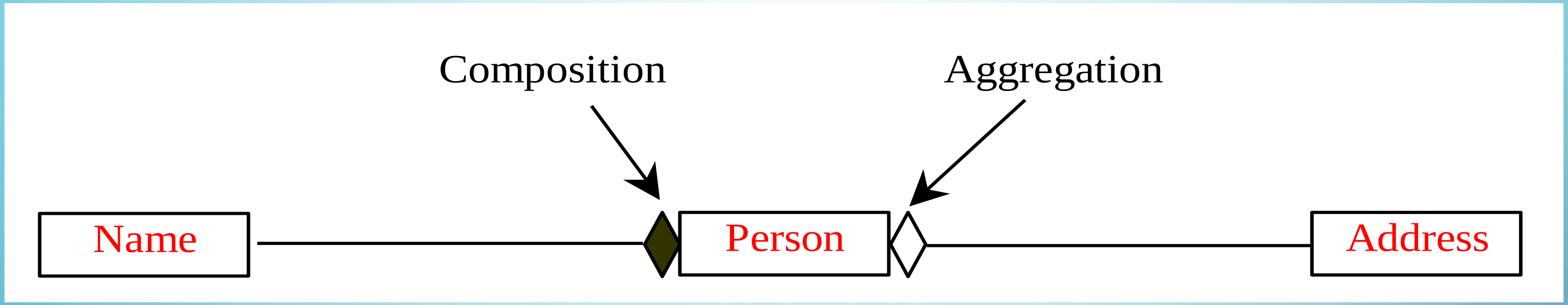
2.1 AGGREGATION

- It is a special form of the association relationship which represents the ownership between two classes.
- It is sometime called as “Has-A relationship”.
- It is loosely owned by another associated class.

2.2 COMPOSITION

- If an object is exclusively owned by an aggregated object, the relationship between two classes will be “Composition”
- It may be called as “Is-A-Part of relationship”.
- It is tightly owned by another associated class.

UML NOTATION



```
public class Name {
  /** Data fields */
  /** Constructors */
  /** Methods */
}
```

```
public class Person {
  /** Data fields */
  private Name name;
  private Address address;

  /** Constructors */
  /** Methods */
}
```

```
public class Address {
  /** Data fields */
  /** Constructors */
  /** Methods */
}
```

INHERITANCE & AGGREGATION

Inheritance	Aggregation
Is-A relationship	Has-A relationship
Geographic🡐Circle	Person<>-Address
If either information hiding and polymorphism are desired, it is recommended.	It give more flexibility because the classes are less dependent.

ABSTRACT & INTERFACE

Abstract	Interface
Strong Is-An-Extension-of relationship	Weak Is-An-Extension-of relationship

Geographic←Circle	Comparable←Circle
	Fruit←Orange

If the relationship clearly describes a parent-child relationship	It give more flexible than abstract because it possess a certain property.