Data Structures and Object-Oriented Design VI

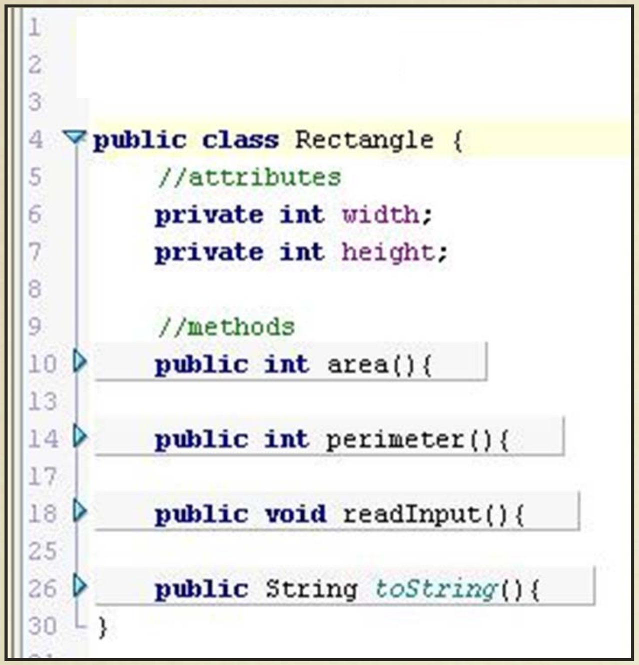
Spring 2014
Carola Wenk

UML

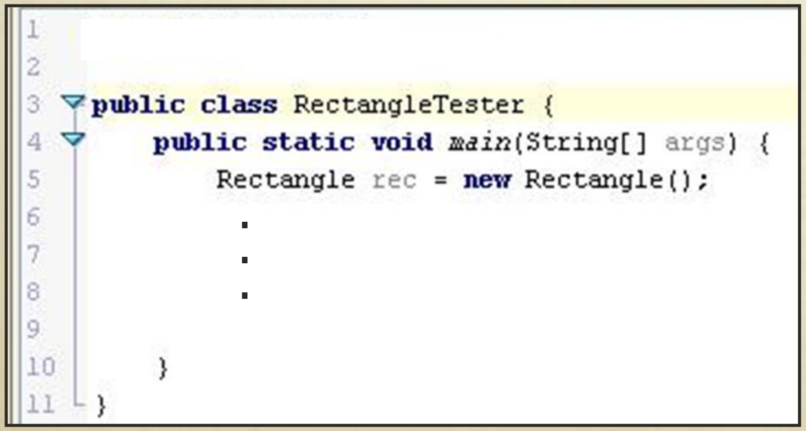
- UML = Unified Modeling Language
- Notation for representing programs in a schematic way
- We will use class diagrams:
    - Show classes
    - Show inter-relationships between classes

Rectangle

Code:

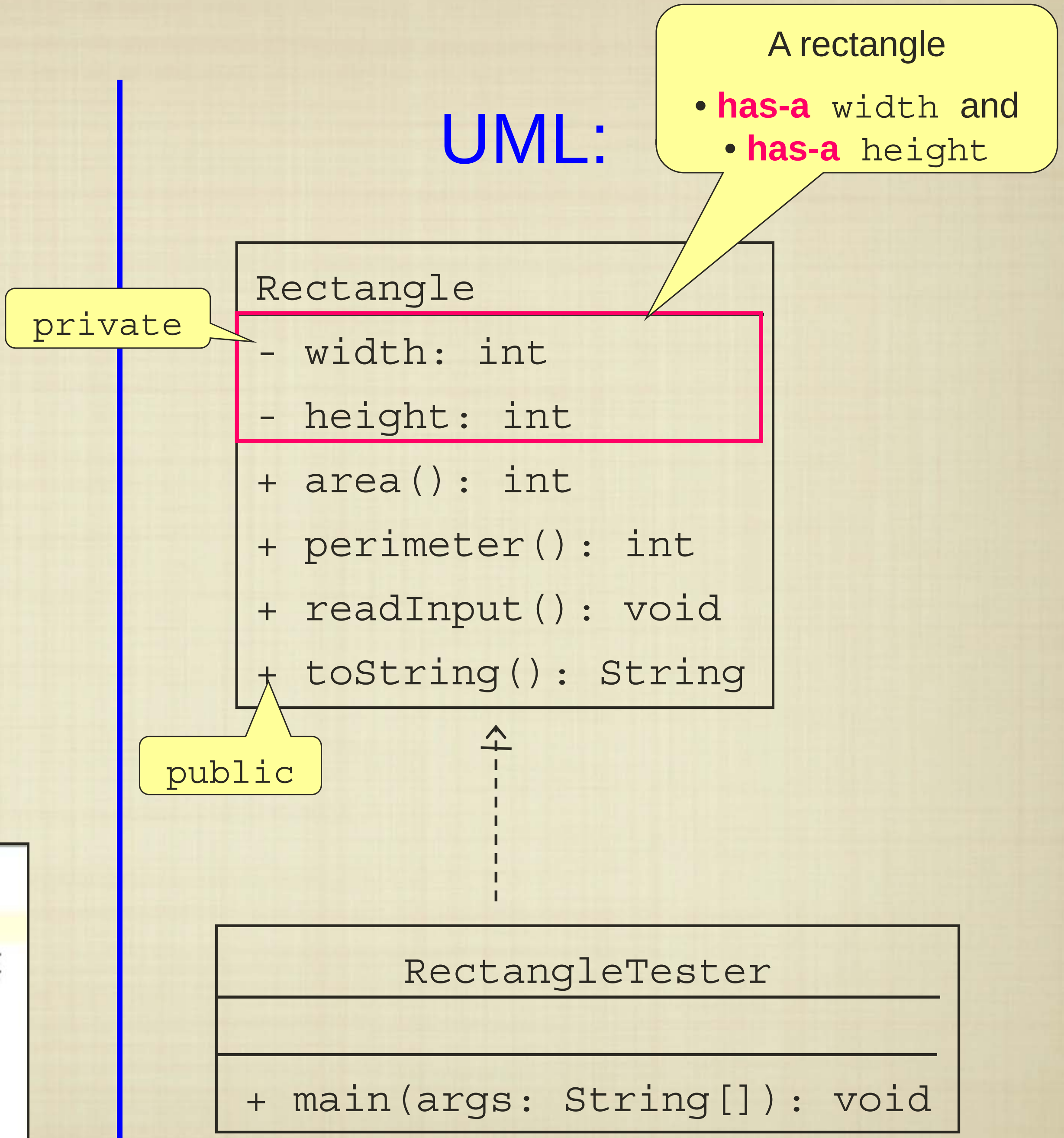


```
public class Rectangle {
    //attributes
    private int width;
    private int height;

    //methods
    public int area(){

    public int perimeter(){

    public void readInput(){

    public String toString(){
}
```



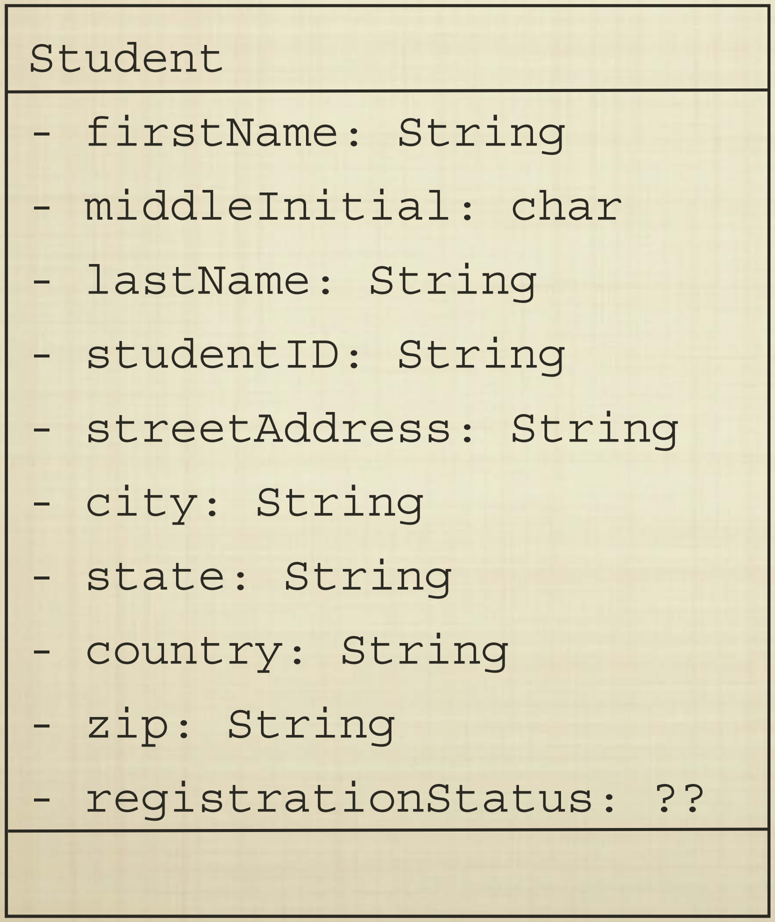
```
public class RectangleTester {
    public static void main(String[] args) {
        Rectangle rec = new Rectangle();
            .
            .
            .

    }
}
```

UML:



Student

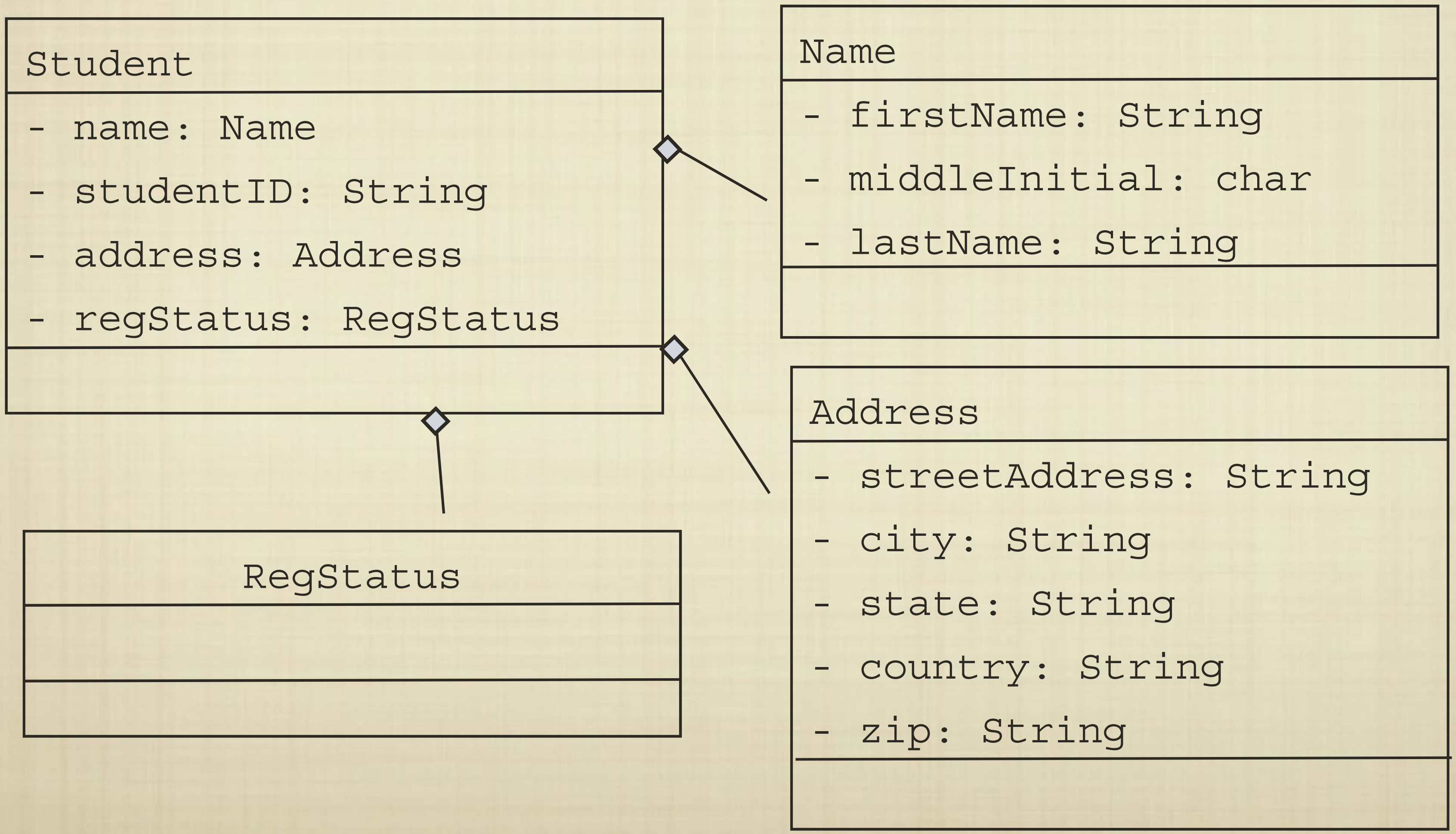
A student is a person enrolled at a university. Each student *has-a* name, a student ID, an address, and a registration status. A student's name consists of a first name , a middle initial and a last name. A student's address consists of a street address, a city, state, country and postal code.



```
Student
- firstName: String
- middleInitial: char
- lastName: String
- studentID: String
- streetAddress: String
- city: String
- state: String
- country: String
- zip: String
- registrationStatus: ??
```

Student

A student is a person enrolled at a university. Each student *has-a* name, a student ID, an address, and a registration status. A student's name consists of a first name , a middle initial and a last name. A student's address consists of a street address, a city, state, country and postal code.



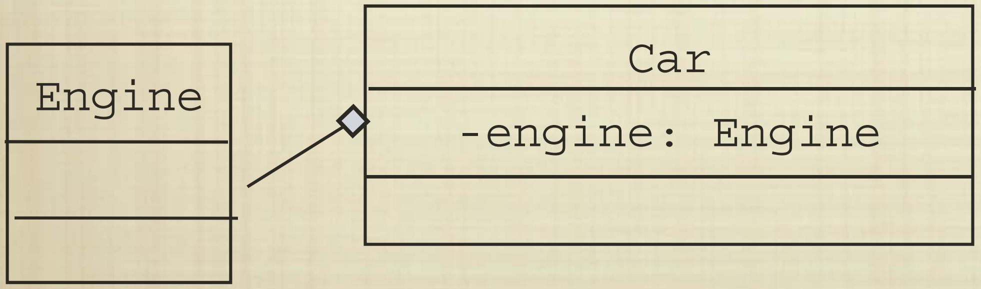
has-a vs. is-a

- **Inheritance** = Derive a new (specialized) class from an existing one

- Relationships in object-oriented design:

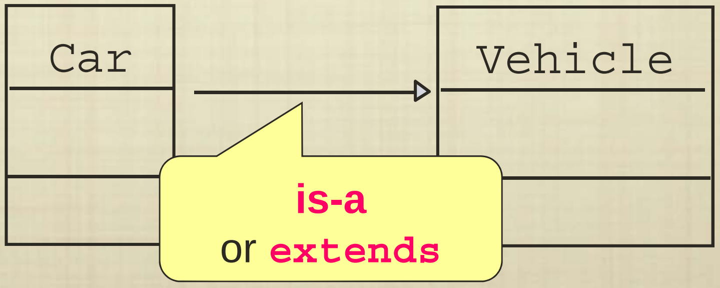
X **has-a** Y

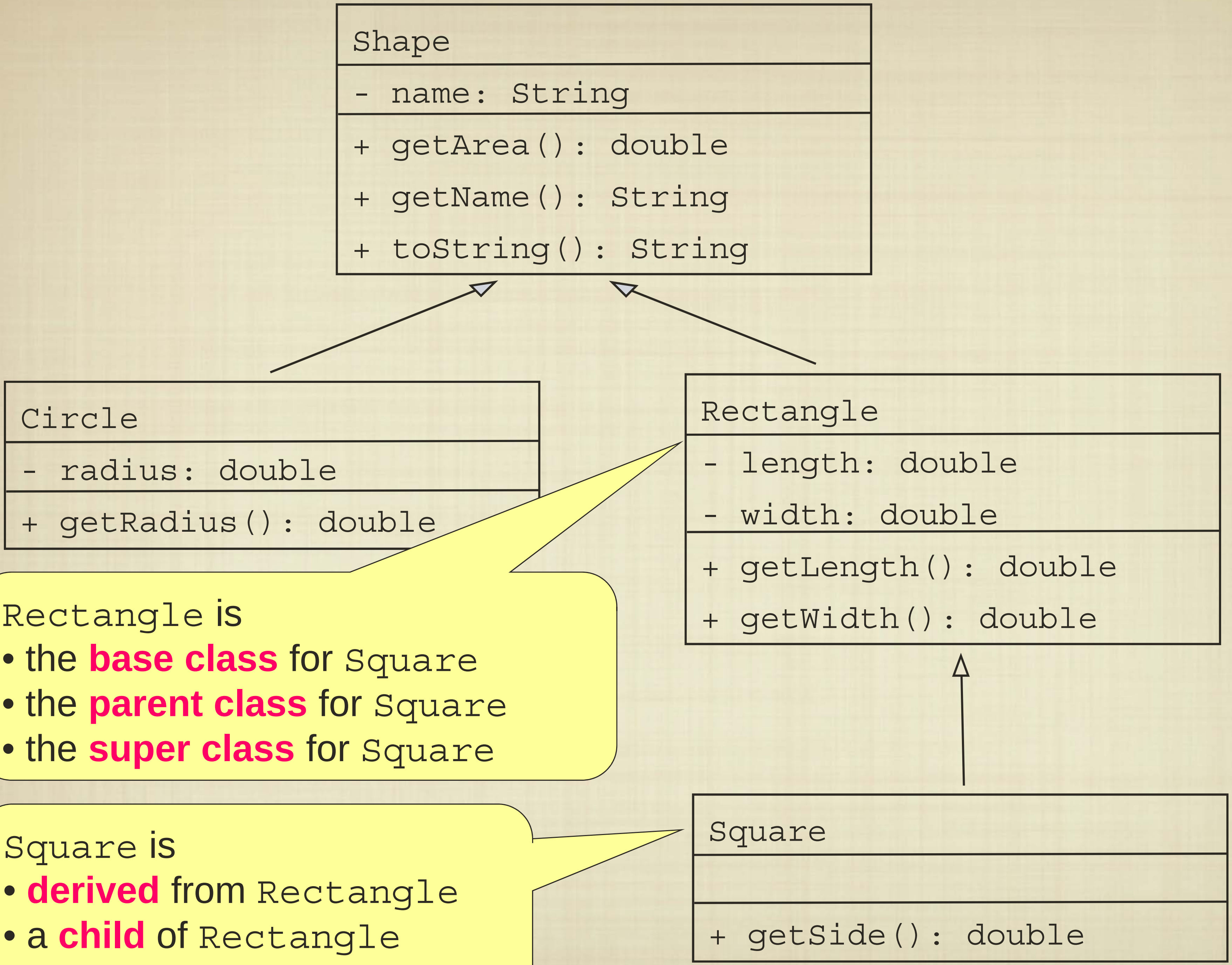
- Y is a component of X
- X and Y are not "the same thing"
- Example: A car **has-a** engine



X **is-a** Y

- X is a specialized instance of Y
- X and Y are "the same thing"
- java: X **`extends`** Y
- Example: A car **is-a** vehicle

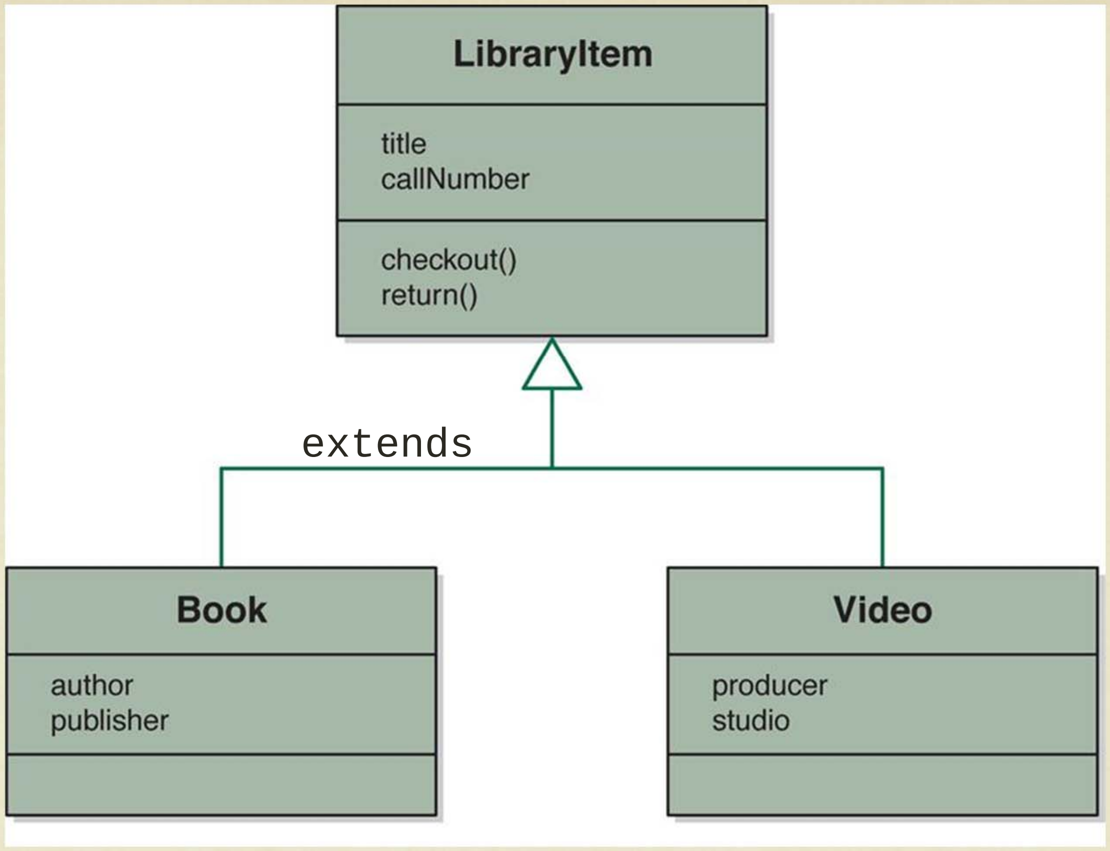




Rectangle is
- the **base class** for Square
- the **parent class** for Square
- the **super class** for Square

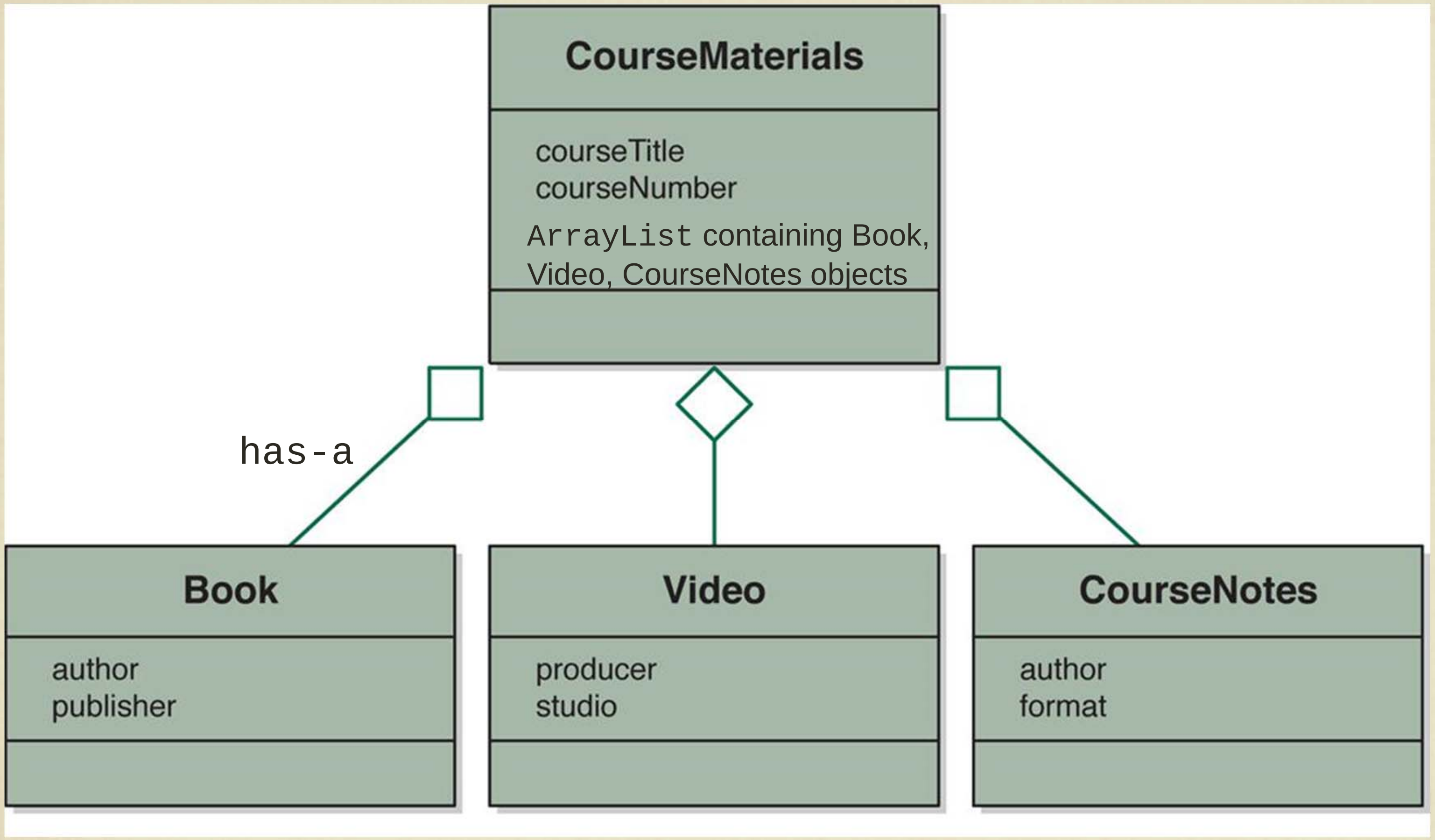
Square is
- **derived** from Rectangle
- a **child** of Rectangle
- a **sub class** of Rectangle

Generalization / Inheritance



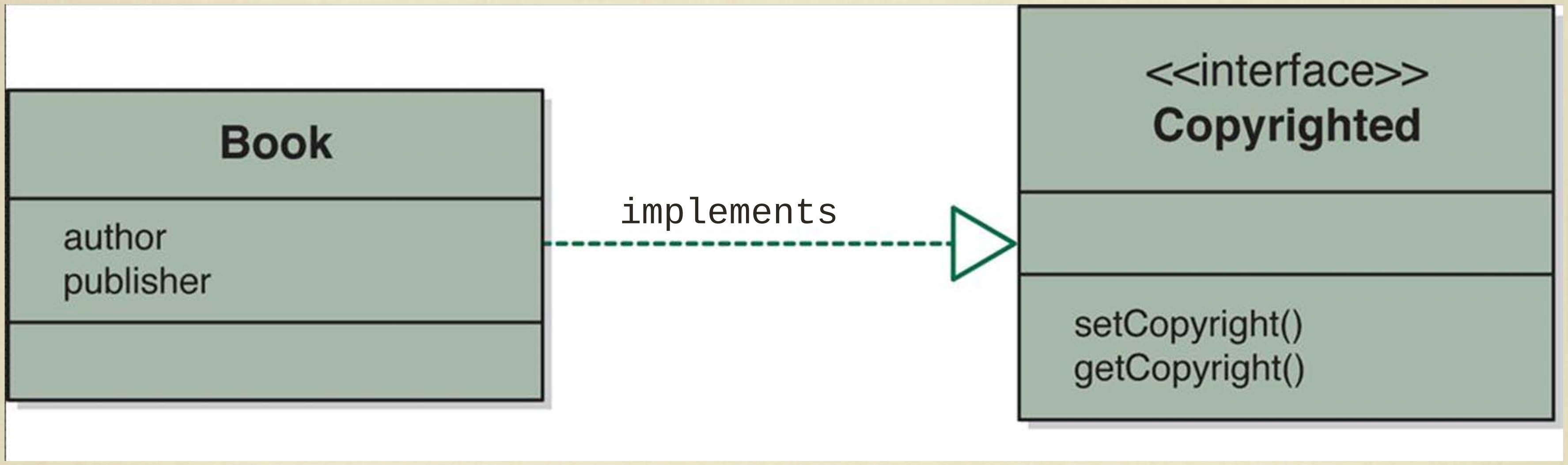
- `extends` is represented by an open arrowhead
- *A* $\longrightarrow\triangleright$ *B* means `A extends B` , or in other words A *is-a* B

Composition



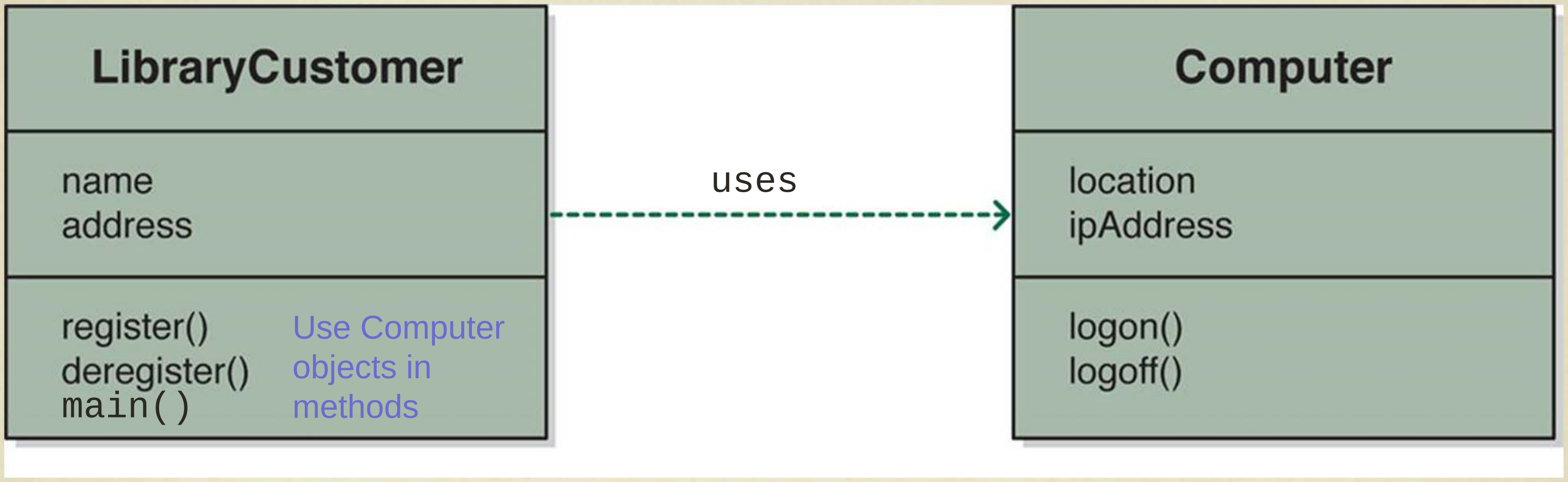
- *has-a* represented by a diamond
- One class contains objects of another class

Interface

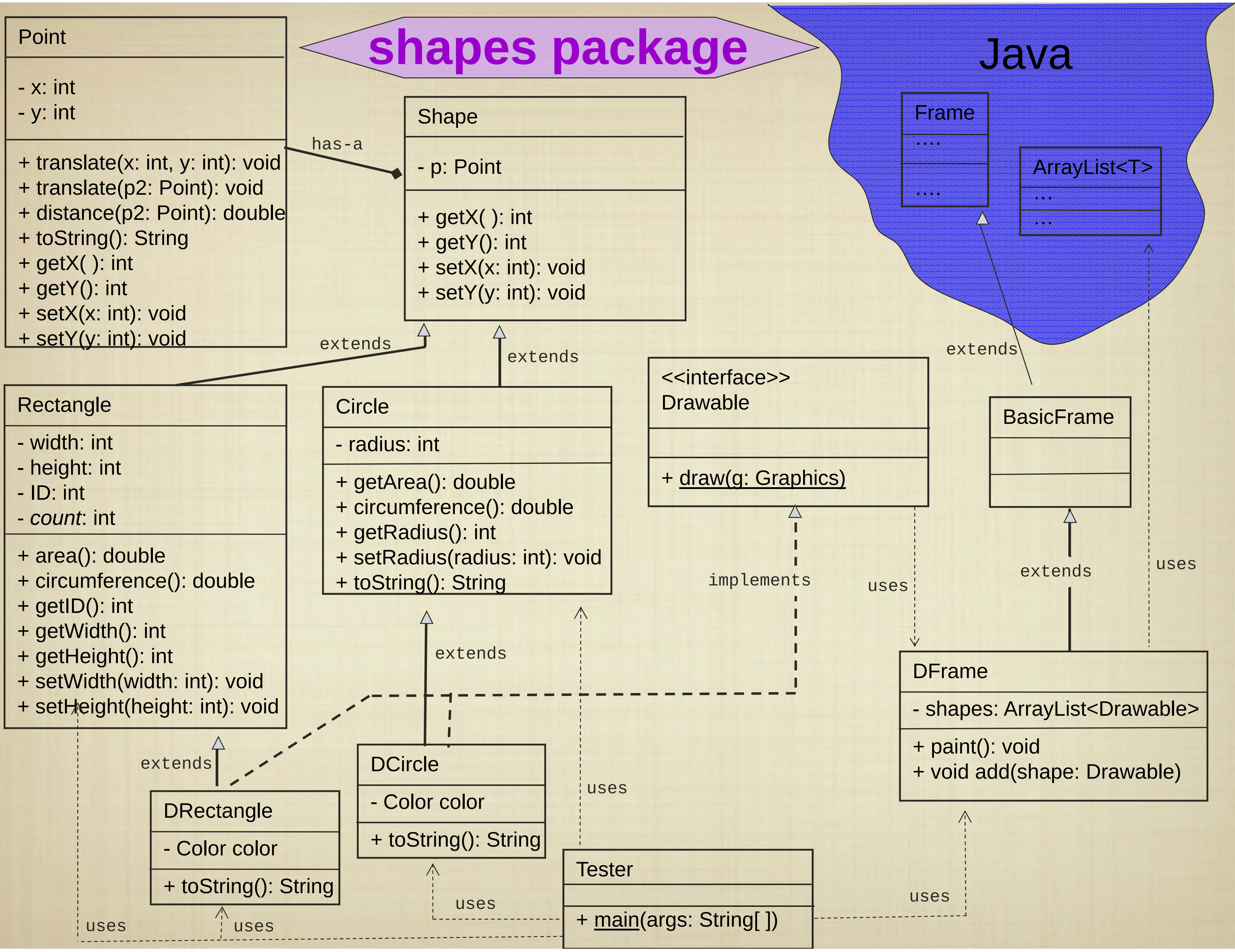


- class A $\dashrightarrow$ interface B means `A implements B`

Dependency / Use



- A $\dashrightarrow$ B means “A uses B”


shapes package
Java
Point
- x: int
- y: int
+ translate(x: int, y: int): void
+ translate(p2: Point): void
+ distance(p2: Point): double
+ toString(): String
+ getX( ): int
+ getY(): int
+ setX(x: int): void
+ setY(y: int): void
has-a
Shape
- p: Point
+ getX( ): int
+ getY(): int
+ setX(x: int): void
+ setY(y: int): void
Frame
....
....
ArrayList<T>
...
...
extends
extends
extends
Rectangle
- width: int
- height: int
- ID: int
- *count*: int
+ area(): double
+ circumference(): double
+ getID(): int
+ getWidth(): int
+ getHeight(): int
+ setWidth(width: int): void
+ setHeight(height: int): void
Circle
- radius: int
+ getArea(): double
+ circumference(): double
+ getRadius(): int
+ setRadius(radius: int): void
+ toString(): String
<<interface>>
Drawable
+ draw(g: Graphics)
BasicFrame
implements
uses
extends
uses
extends
DFrame
- shapes: ArrayList<Drawable>
+ paint(): void
+ void add(shape: Drawable)
extends
DRectangle
- Color color
+ toString(): String
DCircle
- Color color
+ toString(): String
uses
Tester
+ main(args: String[ ])
uses
uses
uses
uses