UNIVERSITY OF TWENTE.

Class hierarchy

Topic of Software Systems (TCS module 2)

Lecturer: Tom van Dijk

OBJECT-ORIENTED PROGRAMMING

Object-oriented programming *so far*

- An object is an instance of a class
- Abstraction: hide details, only make public what is necessary
- Encapsulation: hide information (private fields vs public methods)
- Separation of concerns
- Constructors, initializers, and garbage collection

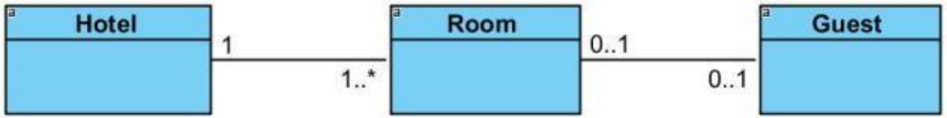
CLASS HIERARCHY

Object-oriented programming *in this topic*

- Class hierarchy
- Subclasses
- Inheritance
- Polymorphism
- DRY principle: Don’t Repeat Yourself

EXAMPLE: HOTEL INFORMATION SYSTEM

- Program design defines relations between concepts (classes)
- Last week: association: “has-a”, “belongs-to”, “knows”



- Now: is-a relation

THE IS-A RELATIONSHIP

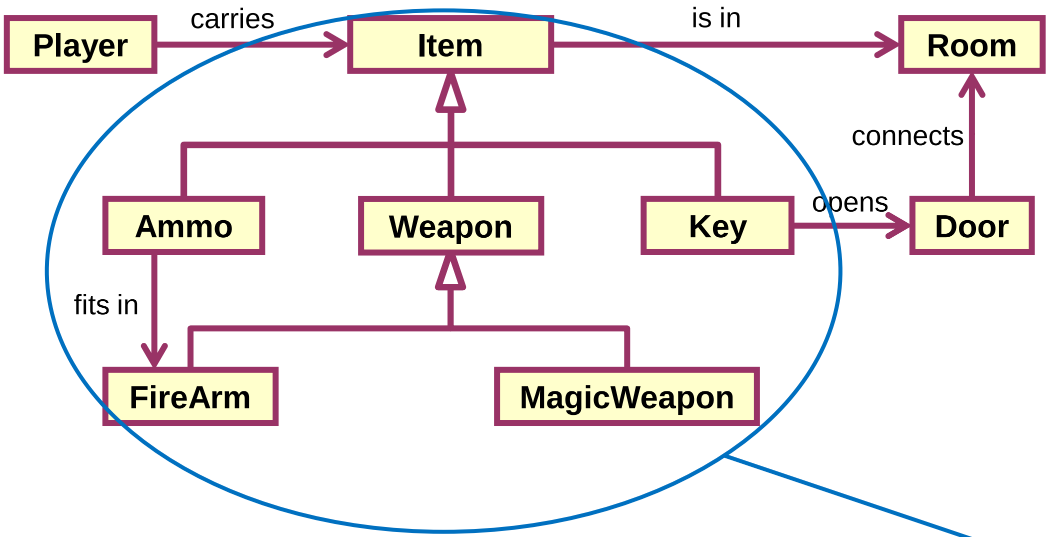
- Sometimes a B is-an A:

- A car is a vehicle; a train is a vehicle; a bike is a vehicle
- A bear is an animal; a cat is an animal; an owl is an animal
- A key is an item; a treasure is an item; a chair is an item

A generalizes B
B specializes A
B extends A
A is a superclass of B

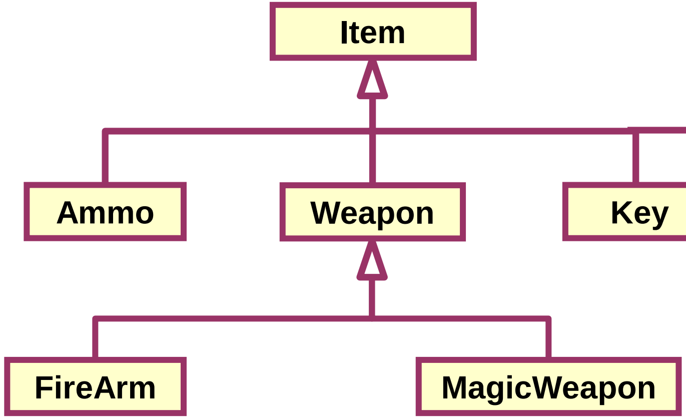
A is an abstraction of B
B inherits from A
B implements A
B is a subclass of A

EXAMPLE: CLASS HIERARCHY IN A MAZE GAME



These is-a relations form a *class hierarchy*

EXAMPLE: WHY USE CLASS HIERARCHIES?



If the game code uses variables of type Item, say, for keeping an Inventory, then it is easy to add more kinds of Items, **without changing the code** for the Inventory**.**

If all Items share common functionality, **only need to program it once** (in Item class)

INHERITANCE

- A subclass inherits fields and methods from its superclass
- Meaning: the child class has the same fields/methods as the parent class
- Principle: Don’t Repeat Yourself (DRY)

- Subclass can now extend the parent class with new fields/methods
- Subclass can now override parent class methods with different method bodies

INHERITANCE IN JAVA

- A subclass inherits fields and methods from its superclass

- Methods **cannot** access private members of the superclass
- Fields hide fields of the superclass with the same name
- Methods override methods of the superclass with the same signature (unless private)

- Fields of a subclass in memory:
  - all fields of the class
  - all fields of the parent class
  - all fields of the parent's parent class, etc.

NEW KEYWORDS

New keyword super functions like this

- Use super to access superclass fields (only if visible!)
- Use super to access original implementations of superclass methods
- Use super(…) to invoke the superclass constructor (only if visible!)

New keyword instanceof: “someObject instanceof C” is true if and only if:

- someObject is an instance of C
- someObject is an instance of a subclass of C (or “of a subclass of a subclass”, etc)

New keyword protected

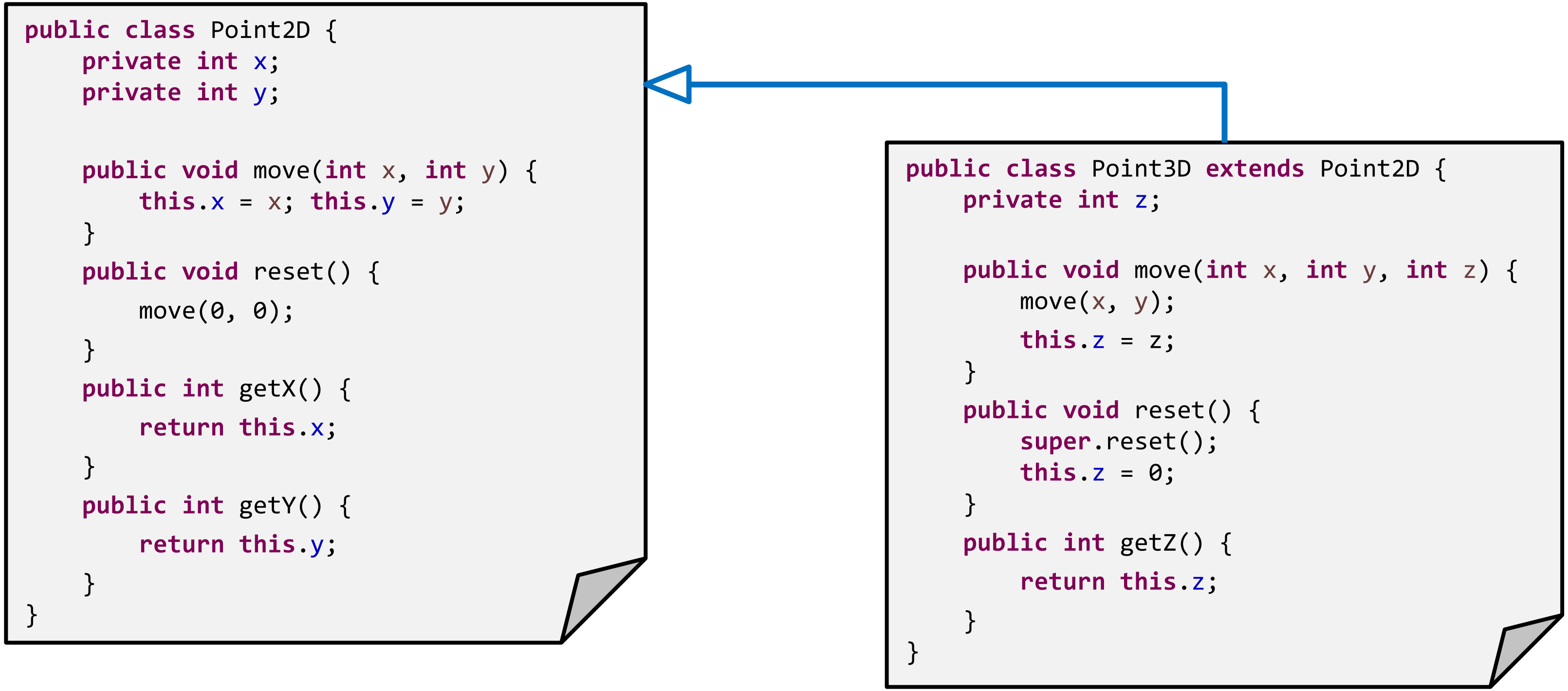
ACCESS MODIFIERS

Fields and methods have an access modifier

Modifier	Same class	Same package	Subclass	The rest
public	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	
(none)	Yes	Yes		
private	Yes			

(*none*) is called: package private

INHERITANCE EXAMPLE



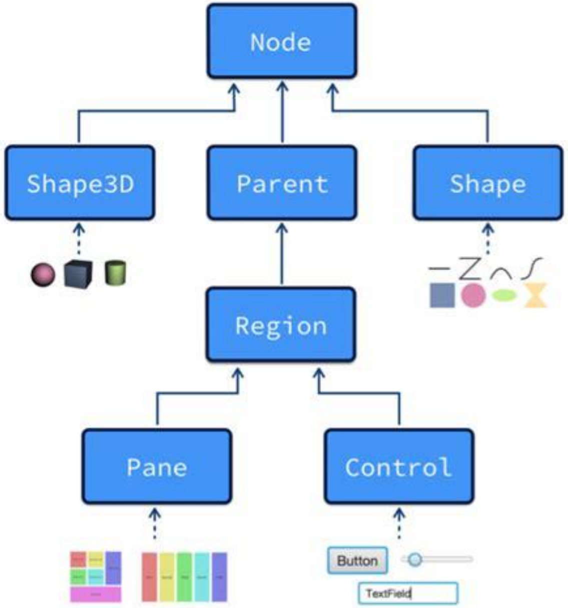
```java
public class Point2D {
    private int x;
    private int y;

    public void move(int x, int y) {
        this.x = x; this.y = y;
    }
    public void reset() {
        move(0, 0);
    }
    public int getX() {
        return this.x;
    }
    public int getY() {
        return this.y;
    }
}
```

```java
public class Point3D extends Point2D {
    private int z;

    public void move(int x, int y, int z) {
        move(x, y);
        this.z = z;
    }
    public void reset() {
        super.reset();
        this.z = 0;
    }
    public int getZ() {
        return this.z;
    }
}
```

EXAMPLE: JAVAFX LIBRARY



(Image from Dzone)

Libraries like JavaFX contain many classes.

Typically, a lot of the “base code” is shared: put in common base classes like Node, or Shape.

For example

- Every Node has a location, rotation, scale
- Every Shape is filled or not filled

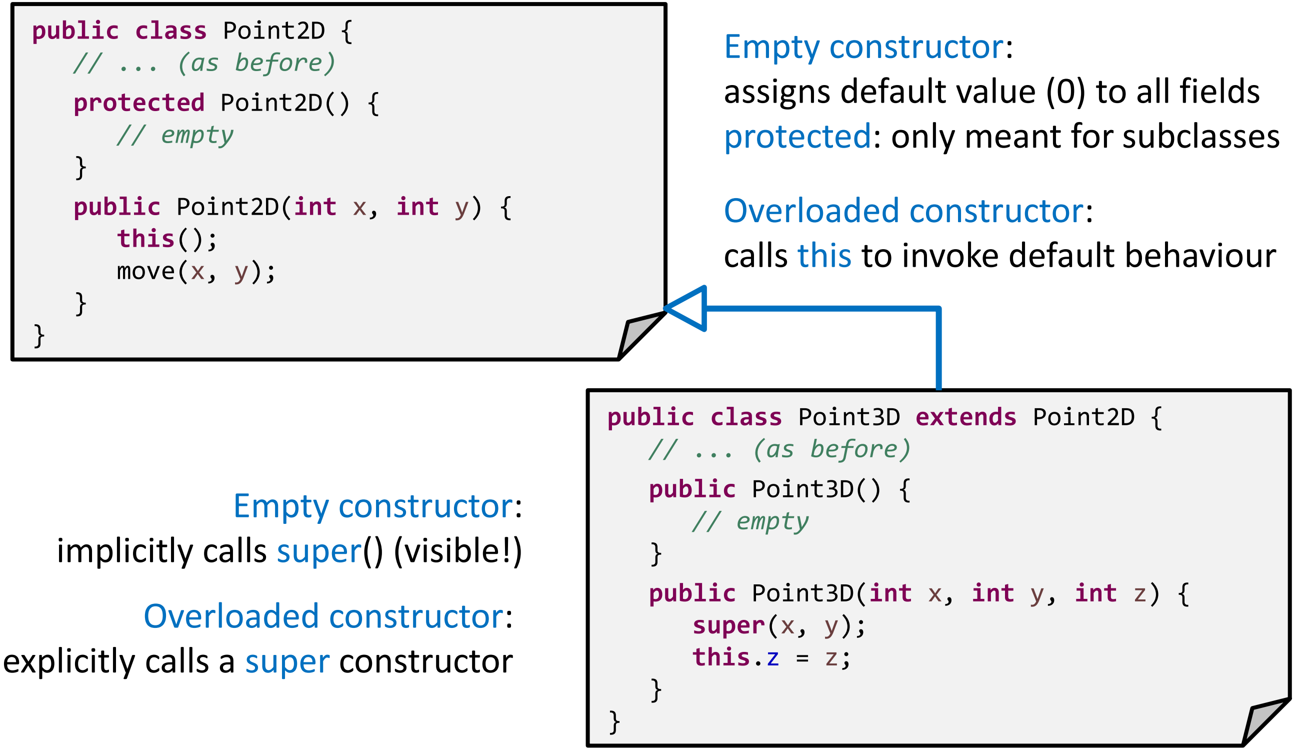
CONSTRUCTORS AND INHERITANCE

- Constructors are not inherited

- Every constructor ***first*** calls the superclass constructor
  - First line: super(args); or this(args);
  - When omitted, by default calls super() (most frequent case)
  - Only valid if superclass has a constructor without parameters!

- Every object has a superclass (by default: inherits from Object)

INHERITANCE EXAMPLE



```java
public class Point2D {
   // ... (as before)
   protected Point2D() {
      // empty
   }
   public Point2D(int x, int y) {
      this();
      move(x, y);
   }
}
```

Empty constructor:
assigns default value (0) to all fields
protected: only meant for subclasses

Overloaded constructor:
calls this to invoke default behaviour

```java
public class Point3D extends Point2D {
   // ... (as before)
   public Point3D() {
      // empty
   }
   public Point3D(int x, int y, int z) {
      super(x, y);
      this.z = z;
   }
}
```

Empty constructor:
implicitly calls super() (visible!)

Overloaded constructor:
explicitly calls a super constructor

POLYMORPHISM

Polymorphism allows the same interface to have different implementations

- Overloading (also called static polymorphism)
  - Methods in a class with the same name but different signature
    (actually: just different number or type of parameters)
  - Get different behaviour by using the method with a different signature

- Overriding (also called dynamic polymorphism)
  - Methods of a subclass with the same signature of a method of the parent class
  - Get different behaviour by using a different subclass

OVERRIDING

- Same signature
- Return type must be the same
- Same or stronger access
  - public → public;
  - protected → public or protected
- static → static; non-static → non-static
- final and/or private methods cannot be overridden

Use @Override annotation in front of overriding method

- Improves maintainability: fewer mistakes! Good practice

VARIABLE HIDING AND SHADOWING

What happens if you reuse a variable name?

- Variable shadowing: local variable with the same name as a class variable
- Variable hiding: field in subclass with same name as field in superclass

INHERITANCE EXAMPLE

```java
public class Item {
    private Room place;

    public Item(Room place) {
        this.place = place;
    }

    public Room getPlace() {
        return this.place;
    }

    public boolean isPortable() {
        return false;
    }
}
```

```java
public class Key extends Item {
    private Door door;

    public Key(Room place, Door door) {
        super(place);
        this.door = door;
    }

    @Override
    public boolean isPortable() {
        return true;
    }

    public boolean opens(Door door) {
        return this.door.equals(door);
    }
}
```

CONTRACTS FOR OVERRIDING METHODS

Contract in supertype: general, weak enough to allow overriding

```
public interface ClosedFigure {
   /*@ ensures \result > 0; */
   public int circumference();
}
```

Specialised contract in subtype: specific, concrete & stronger

- The same or weakened precondition
- The same or strengthened postcondition

```
public class Circle implements ClosedFigure {
   /*@ ensures \result == 2 * Math.PI * radius(); */
   public int circumference() { ... }
}
```

Contract of original method is respected

- Calling `circumference` on a `ClosedFigure` will meet expectations

OBJECT-ORIENTED PROGRAMMING

- An object is an instance of a class in a class hierarchy
- Four concepts
    - Abstraction
    - Encapsulation
    - Inheritance
    - Polymorphism
- Two design principles
    - Separation of concerns
    - Don’t Repeat Yourself (DRY)