

# From Processing 2 Java

## Lecture 2

### Sharing, Hiding, Inheritance and Composition



**TU** / **e**

Technische Universiteit  
**Eindhoven**  
University of Technology

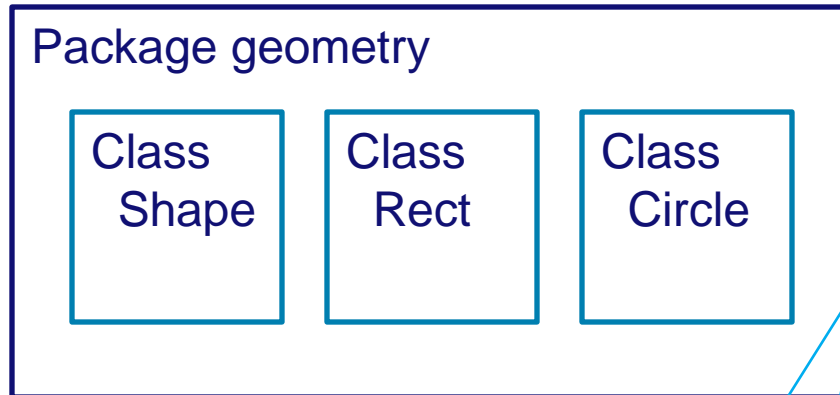
**Where innovation starts**

# Why “Hide / Share” ?

- **Allows for separating functionality from implementation (implementation hiding).**
  - What you *don't* see you *can't* use...
  - What you *do* see you *can* use
- **Allows for creating libraries**
  - “independent” developed pieces of code
  - (possibly) maintained by other people

# Packages

- Packages organise classes in a package
- Class access through `<package>`



- Package defined by `package`
- Package used by `import`

```
package geometry;

public class Shape {
    float x;
    float y;
    float r;
}

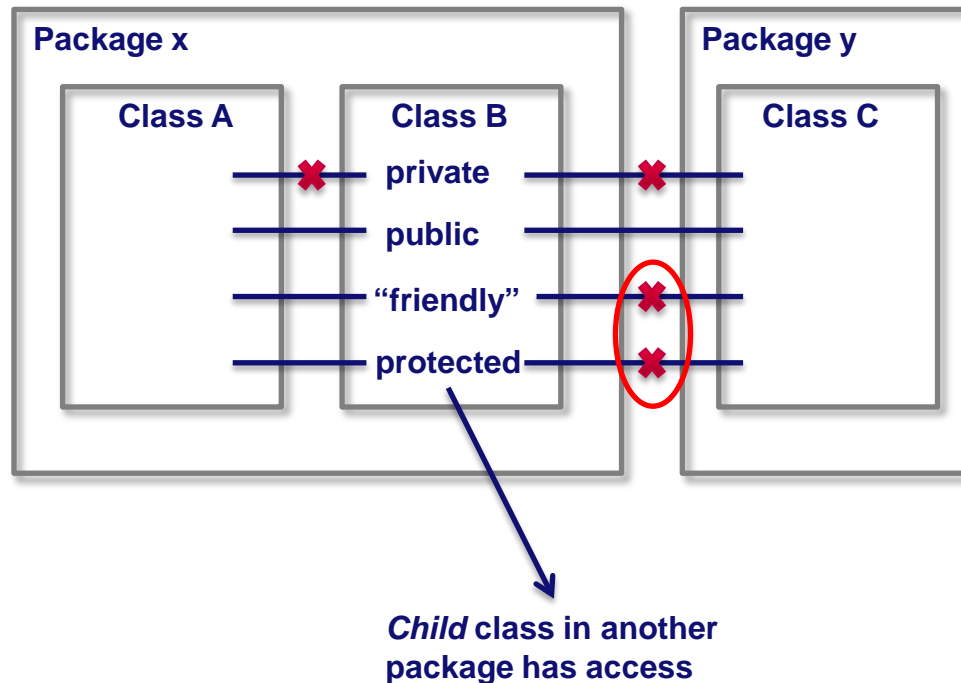
import geometry.*;

class Test {
    Shape a;

    public static void main(String[] args) {
        a = new Shape(0,0,0);
        a.display();
    }
}
```

# Access specifiers

- **public / private / “friendly” / protected**



# Access specifiers

- **public** : everyone
- **protected** : package classes, subclasses and me
- **“friendly”** : package classes and me
- **private** : only me
  
- **Limit access as much as you can...**

# Separating functionality and implementation

- **What you “see” is what you use**
  - **Interface / functionality offered...**
- **What you don’t see you don’t use?**
  - **Methods that help implementation of a class but that are not intended to be used by others.**

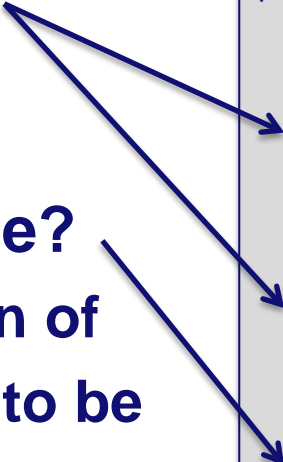
```
package first;

public class dataProcessor {
    private ArrayList dataArray;
    ....

    public void addElement(Object x,
        sortData());
    ....
}

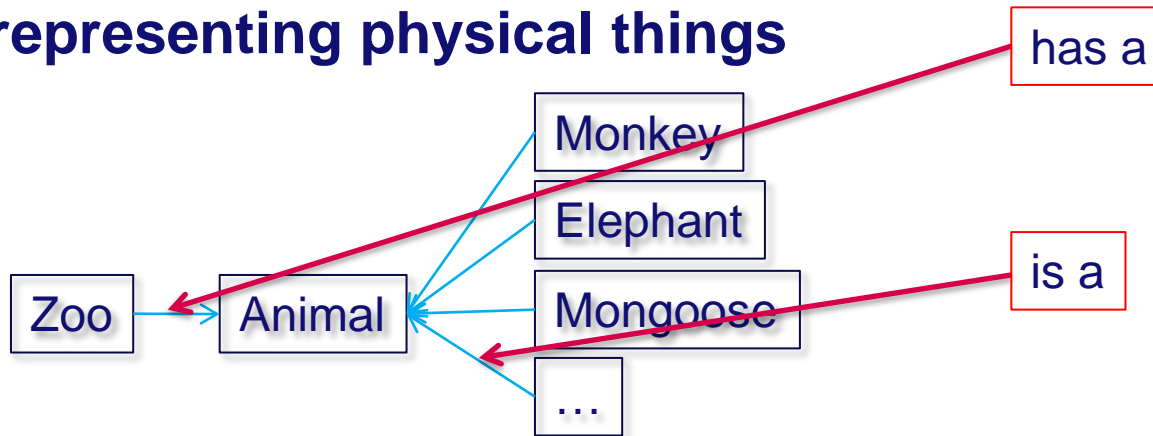
public Object getElement() {
    ....
}

private void sortData() {
    ....
}
}
```

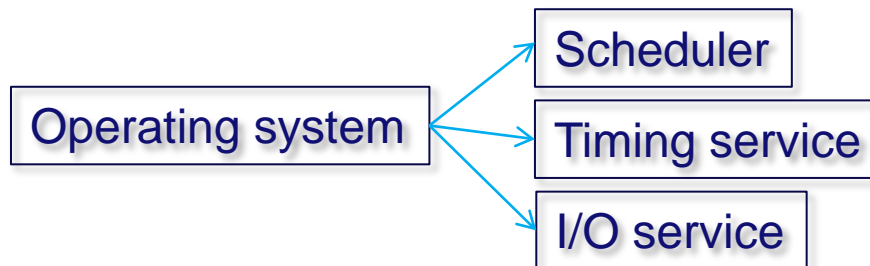


# What about this OO ?

- Think in *objects*
  - representing physical things



- representing abstract things



# Inheritance

- **keyword** : **extends**

```
package nl.tue.id.pp;

public class Shape {
    private int x; // x position
    private int y; // y position
    ....
    public void drawMe() {
        // what to do here?
    }
}
```

```
package nl.tue.id.pp;

class Rectangle extends Shape {
    private int w; // rectangle width
    private int h; // rectangle height

    public void drawMe() {
        ....
    }
}
```

- create a new class that inherits all things public, and protected (and if in same package, also things “friendly”) **AND** adds new functionality and/or properties.



# Composition

- Using other objects as properties
- **NO inheritance of object properties**

```
package nl.tue.id.pp;

public class Rectangle {
    private int x; // x position
    private int y; // y position
    ....

    public Rectangle(int x, int y, int w, int h) {
        ....
    }

    public void drawMe() {
        ...
    }
}
```

```
import nl.tue.id.pp.*;

public class Myap {
    private Rectangle r;

    public Myap() {
        r = new Rectangle(0,0,10,10);
    }

    public void draw() {
        r.drawMe();
        ...
    }
}
```

# Finalization

- **data**
  - constant by definition or initialization
- **arguments**
  - cannot be changed in method
- **methods**
  - cannot be overridden by subclass
  - allows for inline compilation
  - private methods are inherently final...
- **class**
  - cannot be inherited

# Live example

- eclipse

# Summary

- **Packages**
  - `package`, `import`
- **Sharing / Hiding**
  - `public` / `protected` / “friendly” / `private`
- **Inheritance**
  - `extends` extending other objects
- **Composition**
  - using other objects
- **Finalization**
  - fixing the value

# Practical issues

- Packages must be named
  - unique names (nl.tue.id.pp.geometry)
- Packages must be found...
  - CLASPATH = .;D:\JAVA\lib;C:\somewhere\geometry.jar
- Packages *must* adhere to a certain structure
  - nl.tue.id.pp → nl\tue\id \pp\...
  - nl/tue/id/pp/...
- Package *default*