EECS 1720
Building Interactive Systems

Lecture 8b :: Encapsulation & Class Relationships

- Class Relationships (HAS-A vs. IS-A)
- Aggregation/Composition vs. Interfaces/Inheritance

YORK UNIVERSITÉ UNIVERSITY

Topics

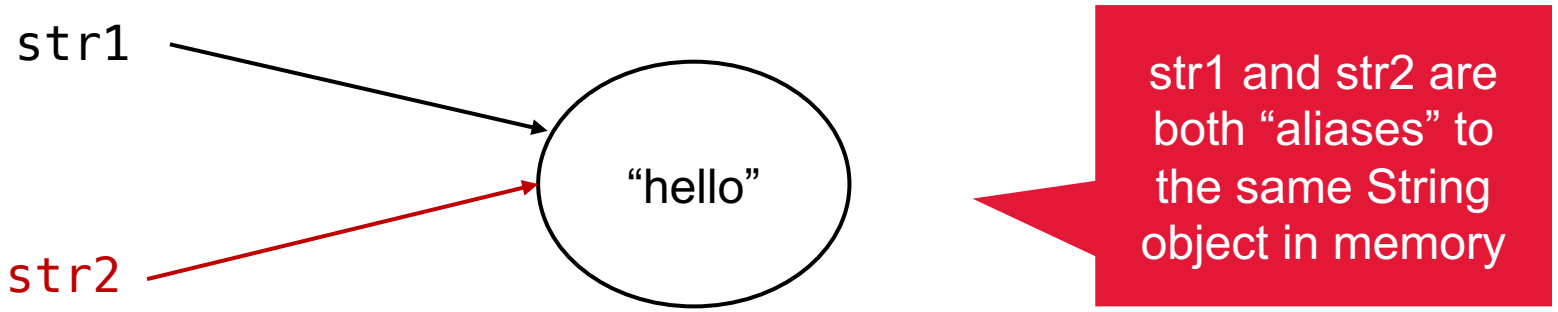
- Class Relationships
  - Has-a (revisited)
    - Aliases vs copies
    - Aggregation vs Composition
  - Is-a
    - Interfaces
    - Inheritance

HAS-A (Aggregation vs. Composition)

Recall: Aliases (from 1710)

- An “alias” is a reference made to an object that already exists in memory and has at least one other existing reference to it

```
String str1 = new String(“hello”);
String str2 = str1;
```

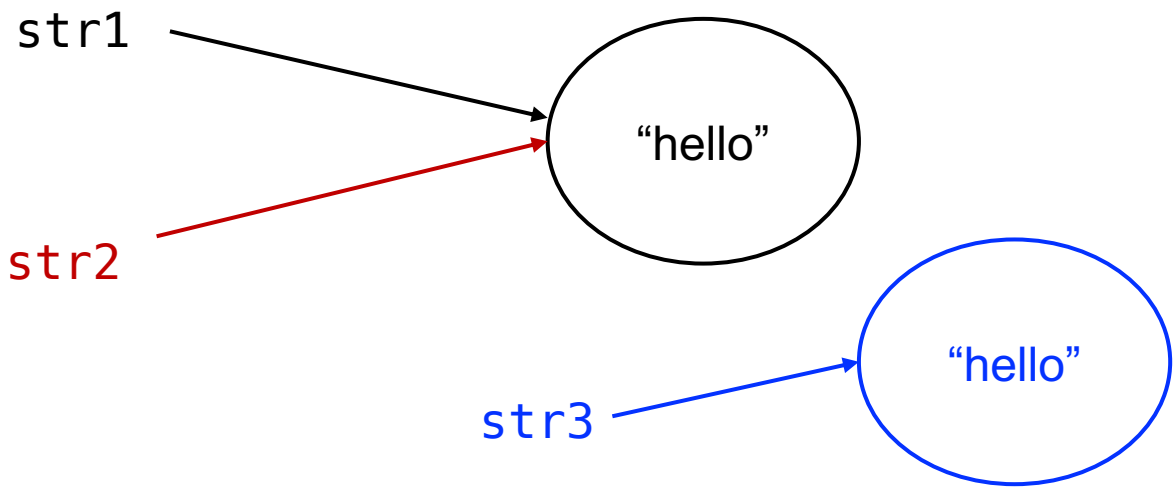


YORK UNIVERSITÉ UNIVERSITY

Copies

- An alias implies no copy (i.e. two or more references refer to the same copy.. thus are aliases of one another)

```
String str1 = new String("hello");
String str2 = str1;
String str3 = new String(str1);   // copy of str1
```



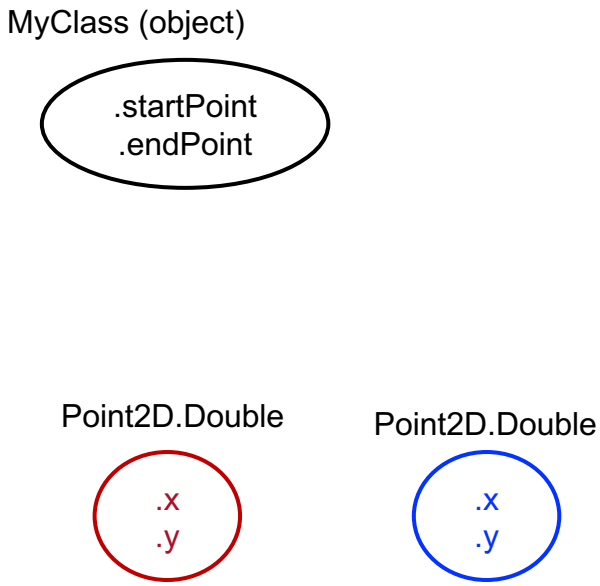
Constructing an object (of a class with fields that are other objects):

- 2 approaches (use **aliases**, or use copies):

```
MyClass
```

```
// fields
-  startPoint : Point2D.Double
-  endPoint : Point2D.Double
```

```
// constructor
+ MyClass(Point2D.Double,Point2D.Double)

// methods
+ getStartPoint() : Point2D.Double
+ setStartPoint(Point2D.Double) : void
```



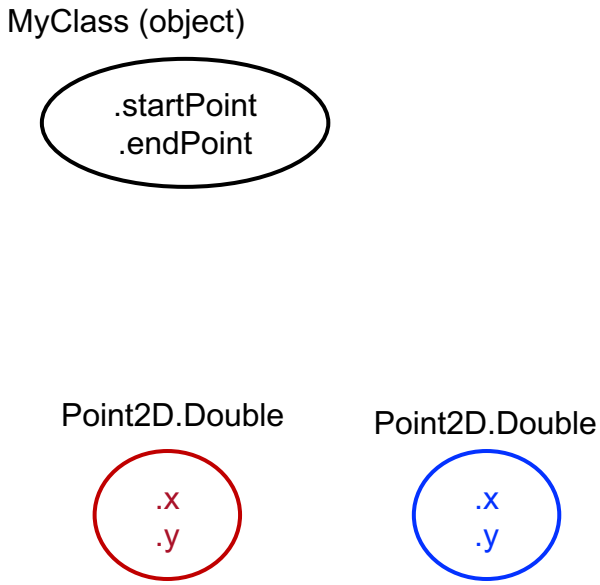
YORK UNIVERSITÉ UNIVERSITY

Constructing an object (of a class with fields that are other objects):

- 2 approaches (use aliases, or use **copies**):

MyClass

```
// fields
-  startPoint : Point2D.Double
-  endPoint : Point2D.Double

// constructor
+ MyClass(Point2D.Double,Point2D.Double)

// methods
+ getStartPoint() : Point2D.Double
+ setStartPoint(Point2D.Double) : void
```



Aggregation vs. Composition

- Aggregation → **uses "aliases"**
  - in a constructor:
    - When initializing a class field (using a reference argument)
    - The argument is simply assigned to the field
  - in an accessor (method):
    - The class field (reference) is returned directly
  - In a mutator (method)
    - A reference argument is assigned directly to the class field

- Significance?
  - If aggregation is used, then the association is "loose" or "weak"
  - The class with the reference fields does not necessarily have exclusive ownership of the objects its fields reference
    - i.e. a client could also have a reference to the same object

Aggregation vs. **Composition**

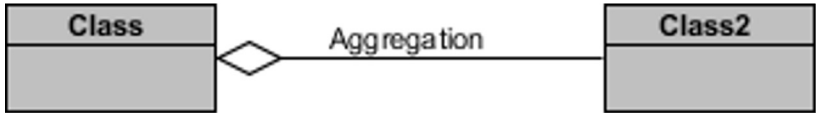
- Composition → **uses "copies"**
  - in a constructor:
    - When initializing a class field (using a reference argument)
    - A copy is created from the argument and the copy is assigned to the class field
  - in an accessor (method):
    - A copy of the class field is made and a reference to this copy is returned
  - In a mutator (method)
    - A copy is created from the argument and the copy is assigned to the class field

- Significance?
  - If composition is used, then the association is "strong"
  - The class with the reference fields has exclusive ownership of the objects its field's reference
    - i.e. the class fields are the ONLY references that exist to these objects
    - If you delete/lose all references to the class object, you will also lose all references to the objects referred to by the class object's fields

Aggregation and Composition

- aggregation implies independence (no ownership)
  - Example:
    - a department is an ***aggregation*** of professors
    - if a department (object) disappears then the professor (objects) that have been aggregated into the department don't disappear
    - department weakly associates professor (objects) together
    - Professor objects exist independently of the department object
- composition implies ownership
  - Example:
    - a university is a ***composition*** of departments
    - if the university (object) disappears then all of its departments (objects) also disappear
    - a department cannot exist without a university
    - the university object OWNS its departments

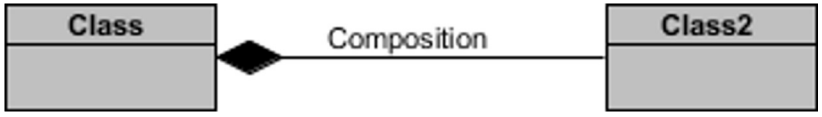
UML representations



Class is an aggregation of Class2

i.e. Instances of Class can maintain references to instances of Class2

(these references are NOT exclusive to Class)



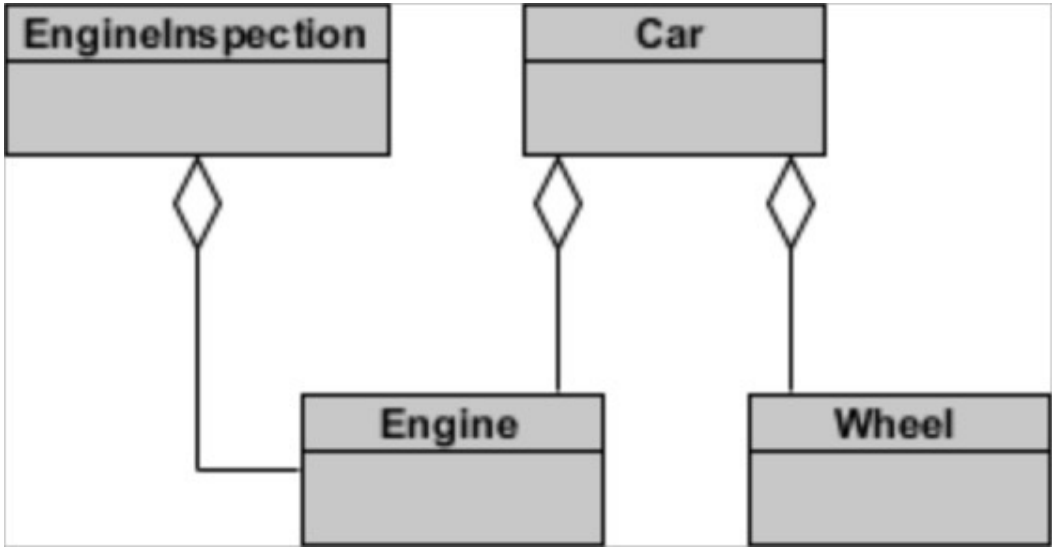
Class is a composition of Class2

i.e. Instances of Class maintain references to isolated copies of instances of Class2

(these references are exclusive to Class)

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-aggregation-vs-composition/

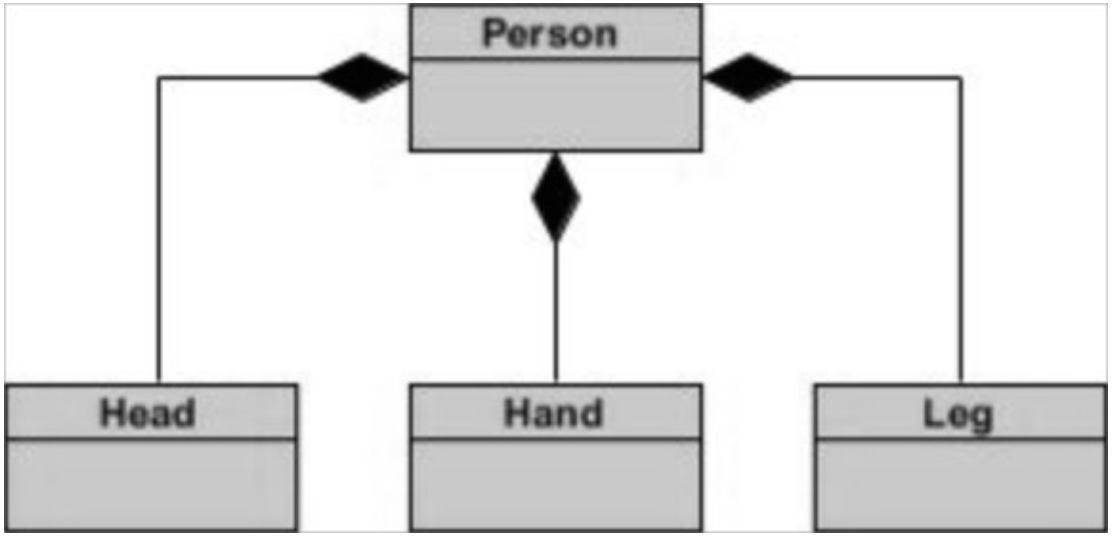
Aggregation vs. Composition (UML)



Generally, the difference is that a class that is an aggregation,
Means it maintains its own ***aliases*** of these fields
(i.e. does not make a copy, only assigns reference)

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-aggregation-vs-composition/

Aggregation vs. **Composition** (UML)



Generally, the difference is that a class that is a composition,
Means it maintains its own ***copies*** of these fields

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-aggregation-vs-composition/

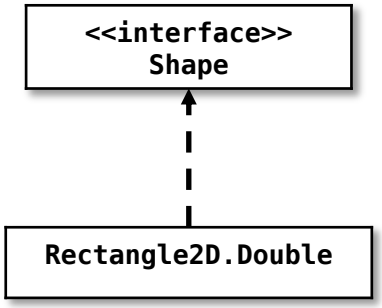
IS-A Relationships

Hierarchical relationships between classes/objects

2 types of “IS-A” relationship

Interfaces

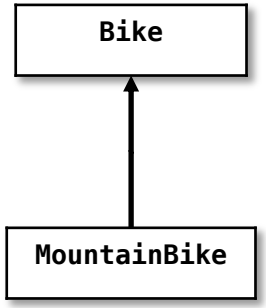
- hierarchical relationship between a special (interface) type and classes



Rectangle2D.Double “is-a” Shape

Inheritance

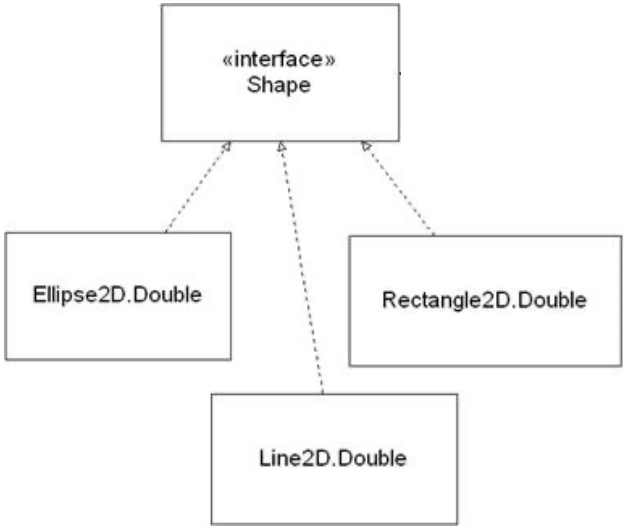
- hierarchical relationship between classes



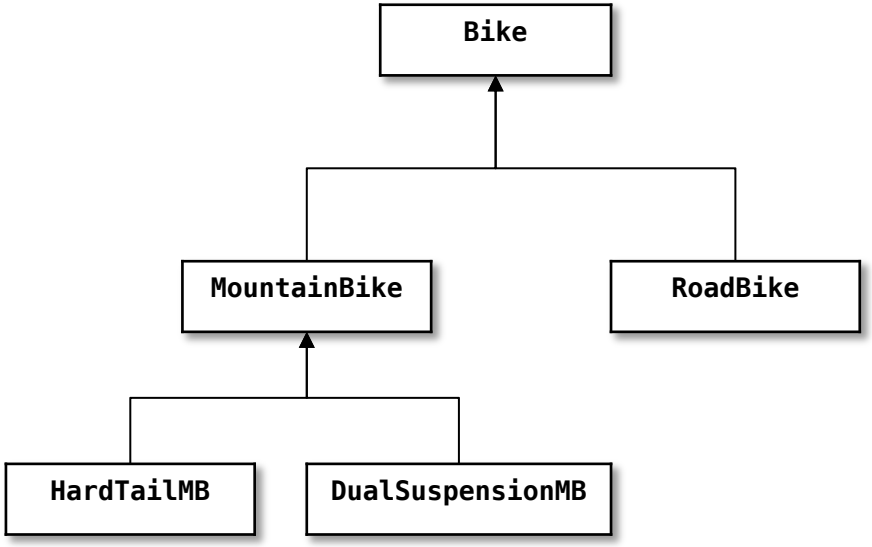
MountainBike “is-a” Bike

What is meant by “IS-A” ?

Interfaces



Inheritance

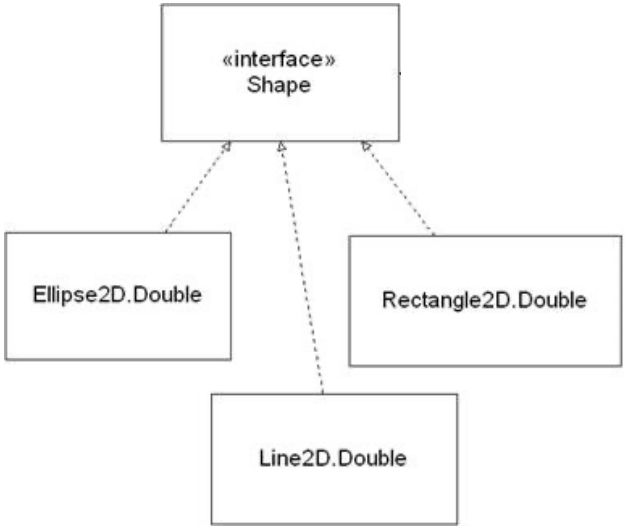


Is-a = “is substitutable for”

YORK UNIVERSITÉ UNIVERSITY

What is meant by “IS-A” ?

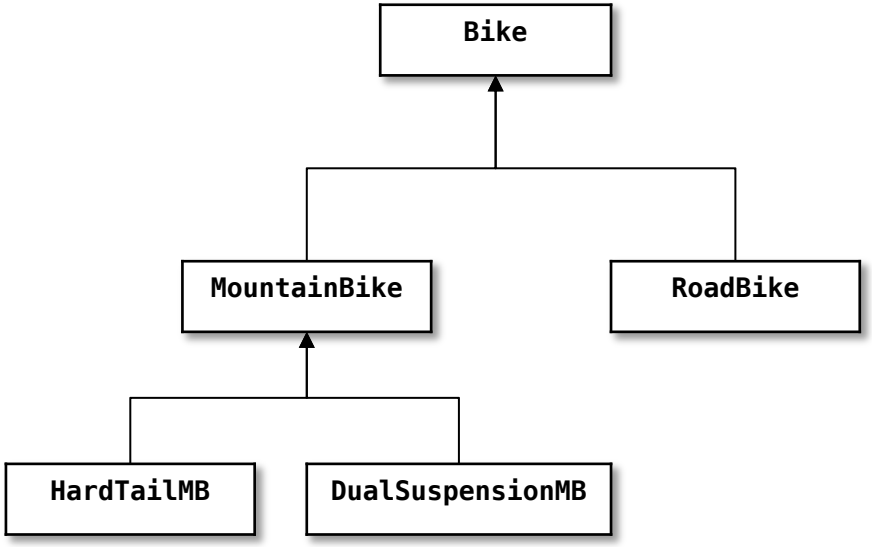
Interfaces



Shape references can be assigned :

Rectangle2D.Double objects
Ellipse2D.Double objects
Line2D.Double objects

Inheritance

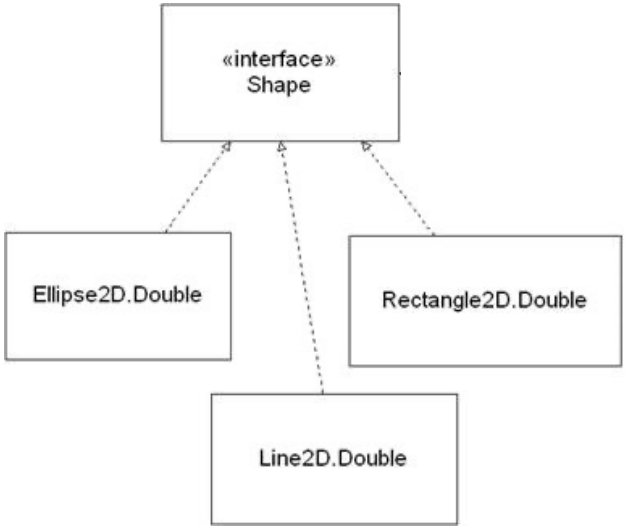


Bike references can be assigned:

Bike, MountainBike, RoadBike,
HardTailMB or DualSuspensionMB objects

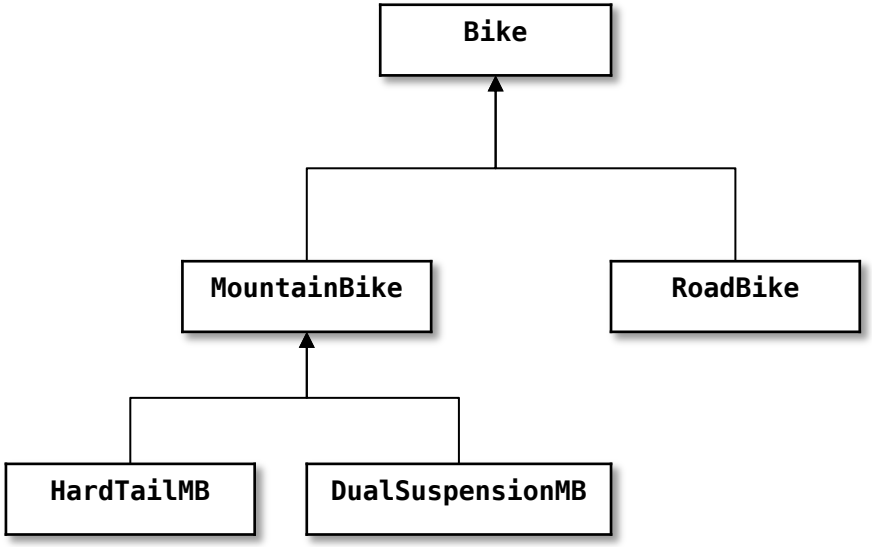
What is meant by “IS-A” ?

Interfaces



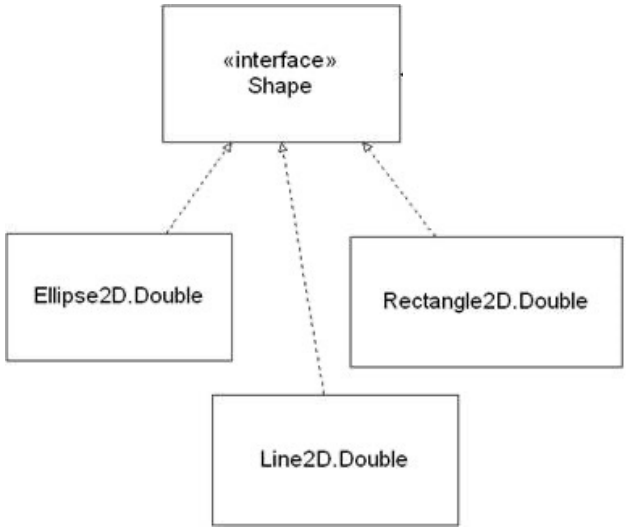
Rectangle2D.Double “is-a” Shape
Ellipse2D.Double “is-a” Shape
Line2D.Double “is-a” Shape

Inheritance



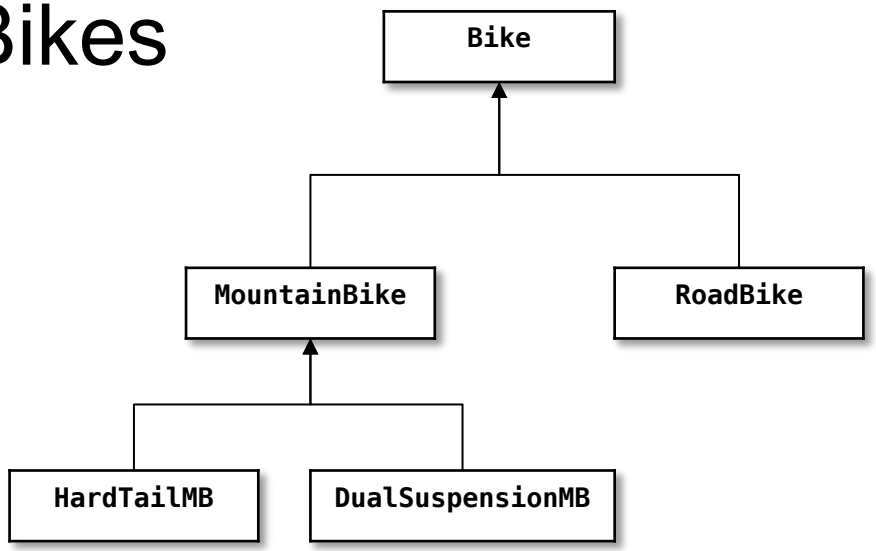
MountainBike ”is-a” Bike
RoadBike “is-a” Bike
HardTailMB “is-a” MountainBike; & “is-a” Bike
DualSuspensionMB “is-a” MountainBike; & ”is-a” Bike

Shapes



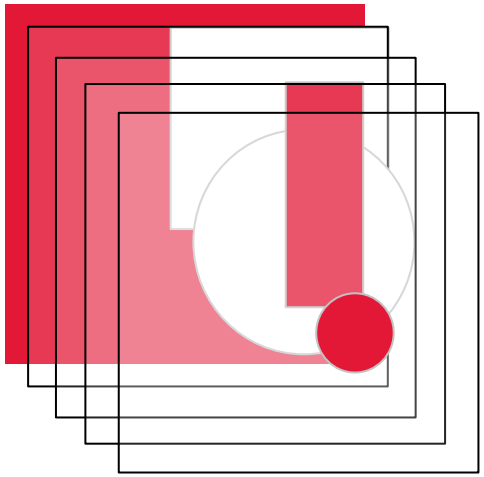

```
// with an interface, this is possible:

Rectangle2D.Double r = new Rectangle2D.Double();
Ellipse2D.Double e = new Ellipse2D.Double();

Shape s;
s = r;
s = e;

// etc

Shape anotherShape = new Line2D.Double();
```

Bikes



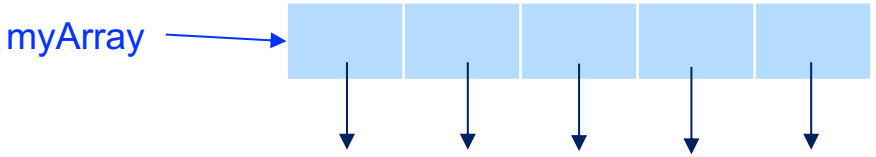

```
// with inheritance, this is possible:

MountainBike mb  = new MountainBike();
RoadBike rb      = new RoadBike();

Bike b;
b = mb;
b = rb;

// etc

Bike anotherBike = new DualSuspensionMB();
```

How would we put the parts of YorkULogo into a single array?



YorkULogo

```
// fields
-  backgnd     : Rectangle2D.Double
-  whiteRect   : Rectangle2D.Double
-  whiteCircle : Ellipse2D.Double
-  redRect     : Rectangle2D.Double
-  redCircle   : Ellipse2D.Double

// constructors & methods (not shown)
```



YORK UNIVERSITÉ UNIVERSITY

```
// inside constructor for YorkULogo, can create
// an array of Shapes  (as arrays must be a single type only)

this.logoParts = new Shape[5];

this.logoParts[0] = this.backgnd;
this.logoParts[1] = this.whiteRect;
this.logoParts[2] = this.whiteCircle;
this.logoParts[3] = this.redRect;
this.logoParts[4] = this.redCircle;
```

logoParts is an array of Shape references

each Shape reference in logoParts array can be assigned any object that has an "is-a" relationship with Shape

YORK UNIVERSITÉ UNIVERSITY

```
// inside constructor for YorkULogo, can create
// an array of Shapes  (array must be of a single type only)

this.logoParts = new Shape[5];

this.logoParts[0] = this.backgnd;
this.logoParts[1] = this.whiteRect;
this.logoParts[2] = this.whiteCircle;
this.logoParts[3] = this.redRect;
this.logoParts[4] = this.redCircle;


// inside drawLogo, can now refer the draw method
// to a Shape type (logoParts[i])

this.gfx.setColor(Color.black);

for (int i=0; i<this.logoParts.length; i++) {

        this.gfx.draw(logoParts[i]);

}
```

Any method that has a Shape argument, can be assigned any object that has an "is-a" relationship with Shape

”is-a” == “is substitutable for”

- A reference type can be **substituted** with any instance of any class that is considered a ”subtype” in that same hierarchy
  - A Shape reference can be substituted with an object of any class that “implements” the Shape interface
  - A Bike reference can be substituted with an object of any class it is considered an ancestor to (i.e. any child, grandchild, etc)

- Similarly, a *method* can have its argument **substituted** with any instance of a class that is considered to have an *“is-a”* relationship with that argument type
  - The draw(..) method accepts a Shape argument, thus we can pass any object of a class that has an “is-a” relationship with Shape

Substitution Principle:

When a parent is expected, a child is accepted