EXPLORE DESIGN PERFECTION

Introduction to Java

Carlos Kavka

ESTECO SpA

EXPLORE DESIGN PERFECTION

Introduction to Java

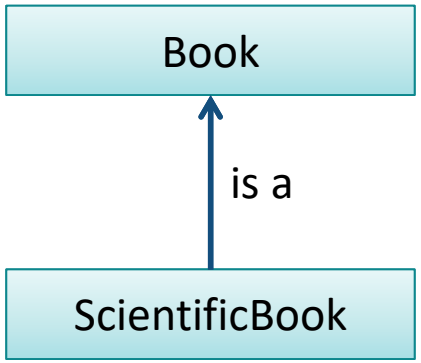
Part III - inheritance

esteco.com

Inheritance

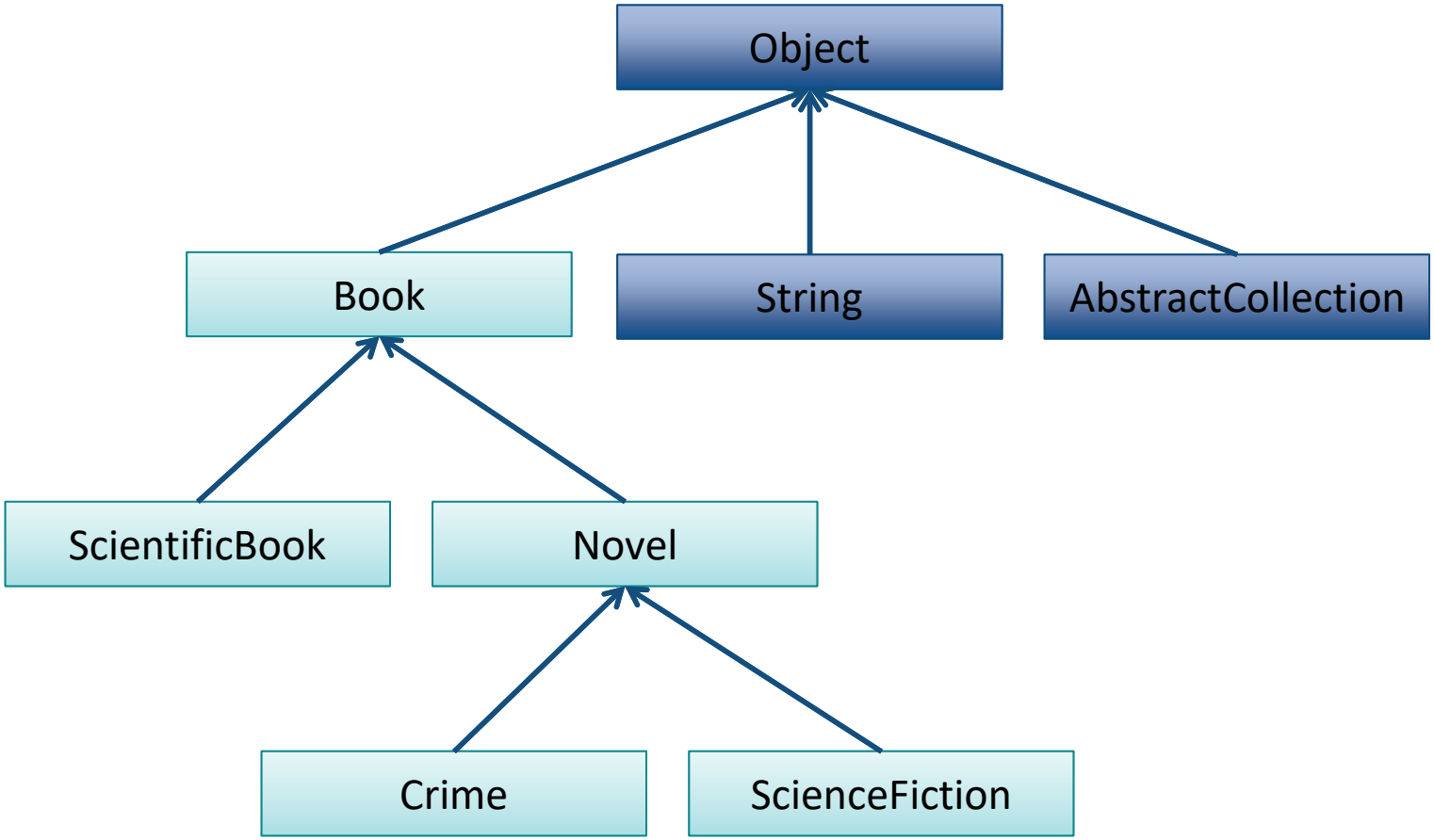
Inheritance allows to define new classes by reusing other classes, specifying just the differences.

It is possible to define a new class (subclass) by specifying that the class must be like other class (superclass)



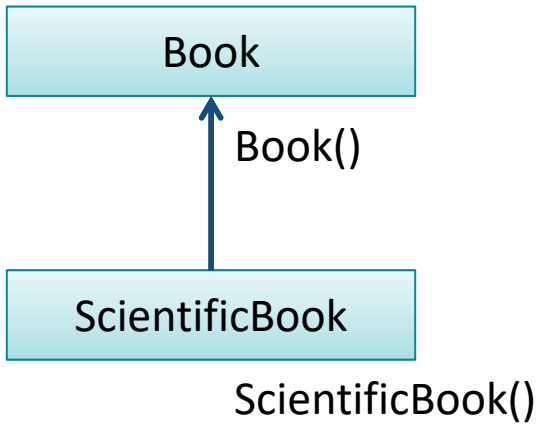

```
public class ScientificBook extends Book {
  String area;
  boolean proceeding = false;
}
```

Inheritance



Constructors definition

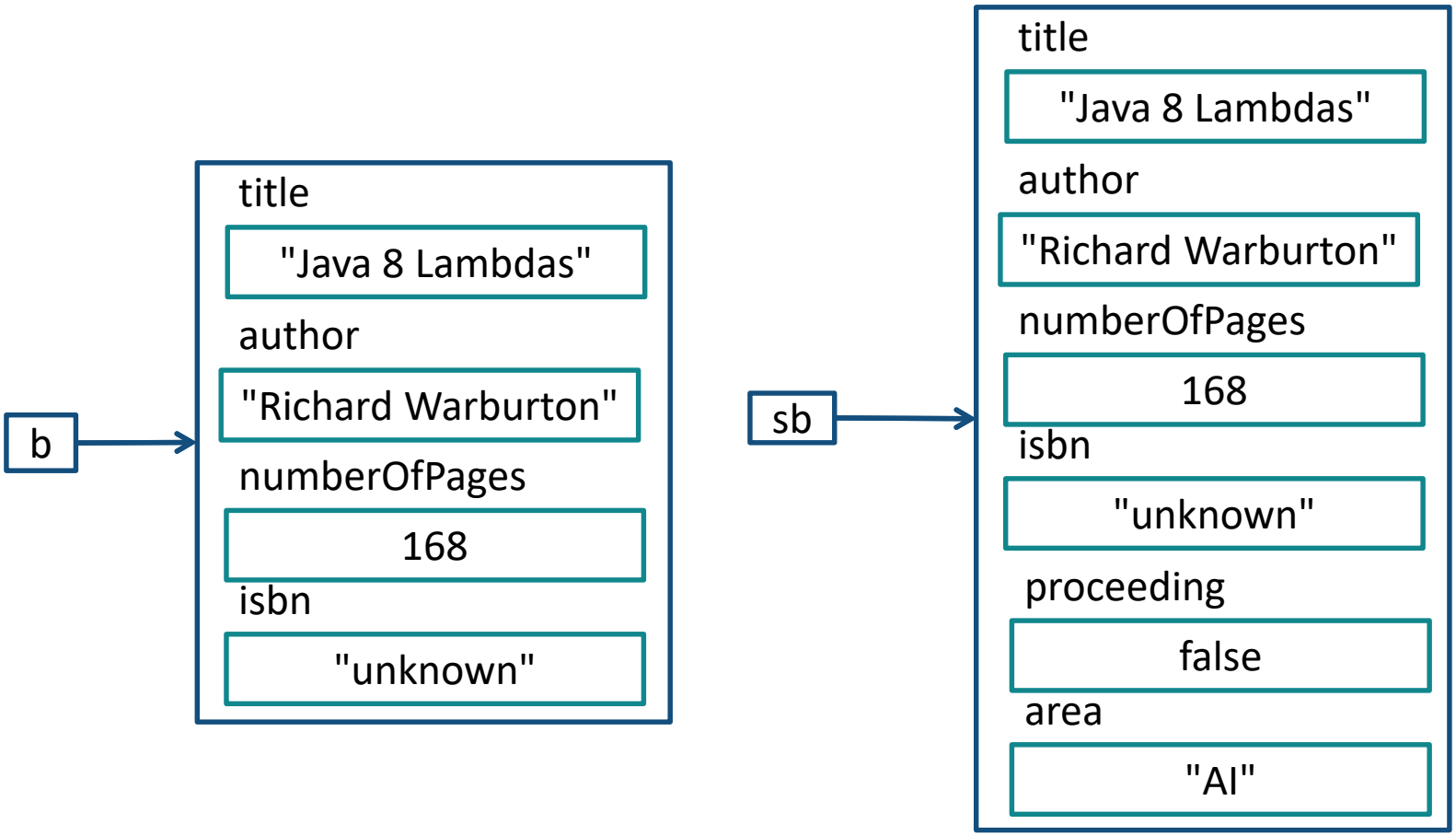
```
public class ScientificBook extends Book {
  String area;
  boolean proceeding = false;

  ScientificBook(String tit, String aut, int num,
                 String isbn, String a) {
    super(tit,aut,num,isbn);
    area = a;
  }
}
```



If the superclass defines a constructor, the subclass has to define it and call the higher one by using super

```
ScientificBook sb;
sb = new ScientificBook("Neural Networks","Simon Haykin",696,"0-02-352761-7","AI");
```

Constructors definition



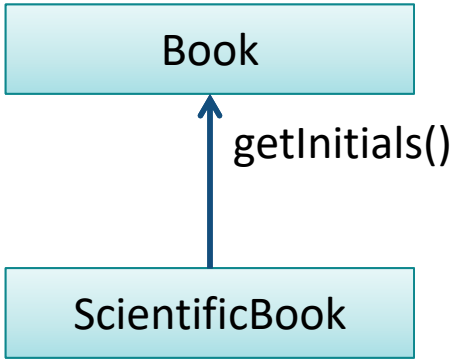

```
ScientificBook sb;
sb = new ScientificBook(" Java 8 Lambdas "," Richard Warburton ",168,
                        "978-1-449-37077-0","AI");
Book b = new Book("Java 8 Lambdas","Richard Warburton",168);
```

Inheritance with methods

- ✓ New methods can be defined in the subclass to specify the behavior of the objects of this class
- ✓ When a message is sent to an object, the method is searched for in the class of the receptor object.
- ✓ If it is not found then it is searched for higher up in the hierarchy.

Inherited methods

Inherited method can be used directly on the instances of the subclass

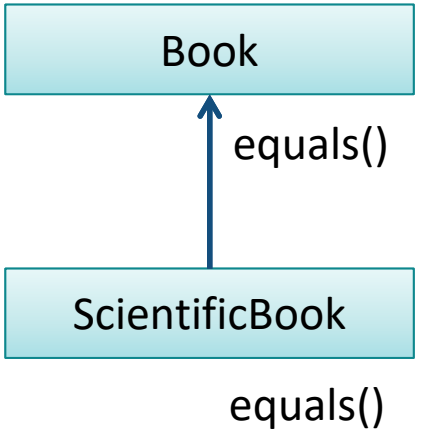



```
ScientificBook sb;
sb = new ScientificBook("Neural Networks","Simon Haykin", 696,
        "0-02-352761-7","AI");
System.out.println(sb.getInitials());
```

Overridden methods

```
public class ScientificBook extends Book {
  String area;
  boolean proceeding = false;

  ScientificBook(String tit, String aut,
      int num, String isbn, String a) {
    super(tit,aut,num,isbn);
    area = a;
  }

  @override
  public boolean equals(ScientificBook b){
    return super.equals(b) && area.equals(b.area) &&
                  proceeding == b.proceeding;
  }
}
```

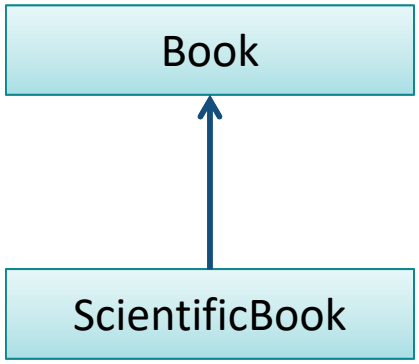
Methods in the subclass can override the methods in the superclass



New methods definition

```
public class ScientificBook extends Book {
   …

  boolean proceeding = false;

   …

public void setProceeding() {
    proceeding = true;
  }

public boolean isProceeding() {
    return proceeding;
  }
}
```

New methods can also be defined



setproceeding()
isProceeding()

Methods: an example

```
public class TestScientificBooks {
  public static void main(String[] args) {
    ScientificBook sb1,sb2;

    sb1 = new ScientificBook("Neural Networks","Simon Haykin",
                      696,"0-02-352761-7", "AI");
    sb2 = new ScientificBook("Neural Networks","Simon Haykin",
                      696,"0-02-352761-7",  "AI");
    sb2.setProceeding();
    ScientificBook.setLocation("Kampar");

    System.out.println(sb1.getInitials());
    System.out.println(sb1.equals(sb2));
    System.out.println(sb1.getLocation());
  }
}
```

```
$ java TestScientificBooks
S.H.
false
ScientificBooks are located in Kampar
```

InstanceOf and getClass()

getClass() returns the runtime class of an object

```
Book b1 = new Book("Java 8 Lambdas","Richard Warburton",168);
   System.out.println(b1.getClass().getName());
```

```
Book
```

instanceof is an operator that determines if an object is an instance of a specified class

```
Book b1 = new Book("Java 8 Lambdas","Richard Warburton",168);
System.out.println(b1 instanceof Book);
```

```
true
```

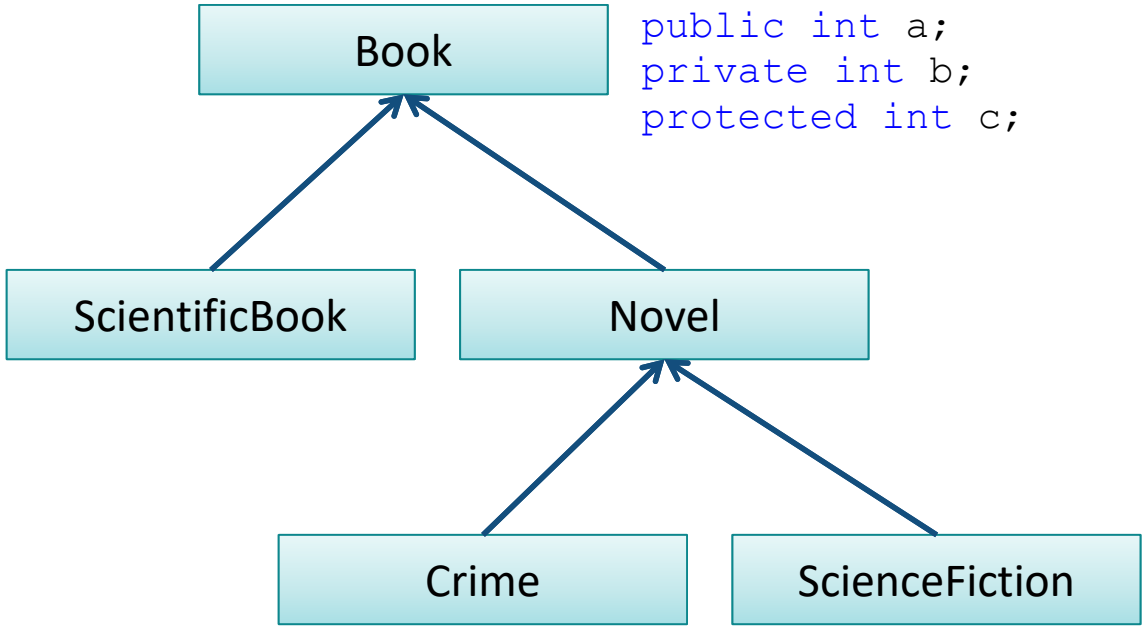
InstanceOf and getClass(): an example

```
public class TestClass {
  public static void main(String[] args) {
    Book b1 = new Book("Java 8 Lambdas","Richard Warburton",168);
    ScientificBook sb1 = new ScientificBook("Neural Networks",
                         "Simon Haykin",696,"0-02-352761-7",
                         "Artificial Intelligence");

    System.out.println(b1.getClass().getName());
    System.out.println(sb1.getClass().getName());
    System.out.println(b1 instanceof Book);
    System.out.println(sb1 instanceof Book);
    System.out.println(b1 instanceof ScientificBook);
    System.out.println(sb1 instanceof ScientificBook);
  }
}
```

```
$ java TestClass
Book
ScientificBook
true true false true
```

Access control

It is possible to control the access to methods and variables from other classes with the modifiers: public, private, protected



Access control

Currently, it is possible to set the proceeding condition of a scientific book in two ways

```
sb1.setProceeding();
```

```
sb1.proceeding = true;
```

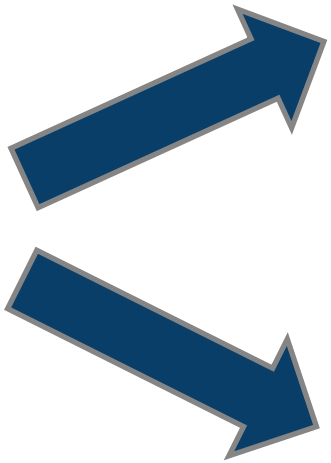
However, direct access to a data member should not be allowed in order to guarantee encapsulation!

```
public class ScientificBook extends Book {
  private boolean proceeding = false;
  …
}
```

```
sb1.setProceeding();      // fine
sb1.proceeding = true;    // wrong
```

Final and abstract

The modifiers final and abstract can be applied to both classes and methods:



A final class does not allow subclassing

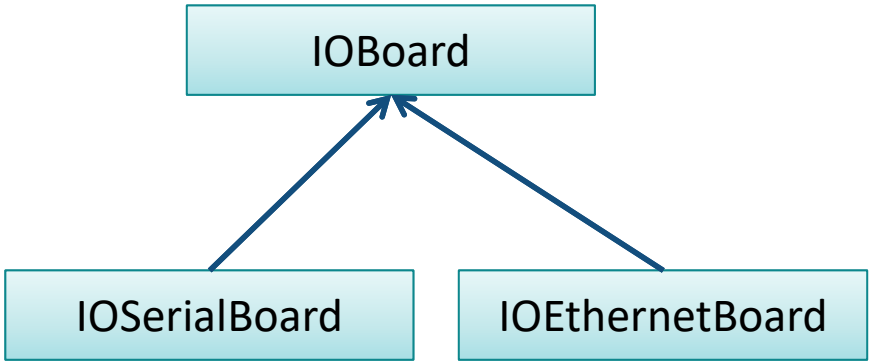
A final method cannot be redefined in a subclass

An abstract class cannot be instantiated

An abstract method has no body, and must be redefined in a subclass

Final and abstract: an example

the class IOBoard and its subclasses



IOBoard is a container for the common behavior of the other boards

Final and abstract: an example

```
public abstract class IOBoard {
  String name;
  int numErrors = 0;

  IOBoard(String s) {
    System.out.println("IOBoard constructor");
    name = s;
  }
  final public void anotherError() {
    numErrors++;
  }
  final public int getNumErrors() {
    return numErrors;
  }
  abstract public void initialize();
  abstract public void read();
  abstract public void write();
  abstract public void close();
}
```

The method anotherError() is final, cannot be redefined in subclasses

The other methods are abstract, subclasses must implement them

Final and abstract: an example

```
public class IOSerialBoard extends IOBoard {
  int port;

  IOSerialBoard(String s,int p) {
    super(s); port = p;
    System.out.println("IOSerialBoard constructor");
  }
  public void initialize() {
    System.out.println("initialize method in IOSerialBoard");
  }
  public void read() {
    System.out.println("read method in IOSerialBoard");
  }
  public void write() {
    System.out.println("write method in IOSerialBoard");
  }
  public void close() {
    System.out.println("close method in IOSerialBoard");
  }
}
```

Final and abstract: an example

```
public class IOEthernetBoard extends IOBoard {
  long networkAddress;

  IOEthernetBoard(String s,long netAdd) {
    super(s); networkAddress = netAdd;
    System.out.println("IOEthernetBoard constructor");
  }
  public void initialize() {
    System.out.println("initialize method in IOEthernetBoard");
  }
  public void read() {
    System.out.println("read method in IOEthernetBoard");
  }
  public void write() {
    System.out.println("write method in IOEthernetBoard");
  }
  public void close() {
    System.out.println("close method in IOEthernetBoard");
  }
}
```

Final and abstract: an example

```
public class TestBoards1 {
  public static void main(String[] args) {
    IOSerialBoard serial = new IOSerialBoard("my first port", 0x2f8);
    serial.initialize();
    serial.read();
    serial.close();
  }
}
```

```
$ java TestBoards1
IOBoard constructor
IOSerialBoard constructor
initialize method in IOSerialBoard
read method in IOSerialBoard
close method in IOSerialBoard
```

Polymorphism

- ✓ It is one of the most important concepts in Object Oriented Programming

- ✓ A solution is polymorphic if the same interface can be used to control a number of different implementations.

- ✓ Example: the power-on interface to request the same operation on a number of very different devices

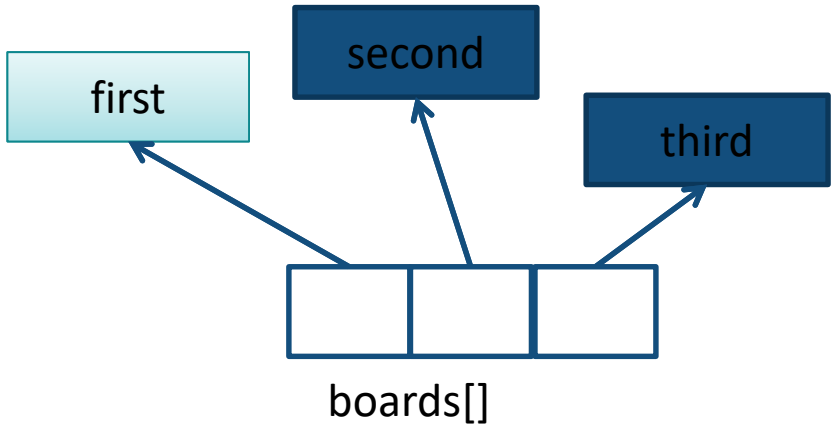
Polymorphism

An array of boards can be defined with IOBoard

```
IOBoard[] board = new IOBoard[3];

board[0] = new IOSerialBoard("my first port",0x2f8);
board[1] = new IOEthernetBoard("my second port",0x3ef8dda8);
board[2] = new IOEthernetBoard("my third port",0x3ef8dda9);
```

Operations are executed based on its corresponding implementation

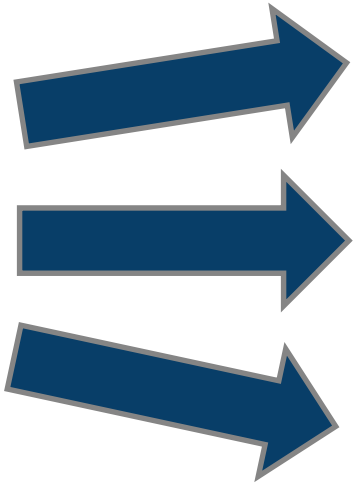
```
for(int i = 0;i < 3;i++)
    board[i].initialize();

for(int i = 0;i < 3;i++)
    board[i].read();

for(int i = 0;i < 3;i++)
    board[i].close();
```



Interfaces

An interface describes what classes should do, without specifying how they should do it.

An interface looks like a class definition where



- All fields are static and final
- there can be abstract methods (public)
- there can be default methods

```
interface IOBoardInterface {
  void initialize();
  void read();
  void write();
  void close();
}
```

```
interface NiceBehavior {
  String getName();
  String getGreeting();
  void sayGoodBye();
  default void sayHello() {
    System.out.println("Hello");
}
```

Interfaces

```
public class IOSerialBoard implements IOBoardInterface, Nice Behavior {
  int port;

  public void initialize() { … }
  public void read() { … }
  public void write() { … }
  public void close() { … }

  public String getName() { … }
  public String getGreeting() { … }
  public void sayGoodBye() { … }

}
```

Note that there is no inheritance

Note a class can implement more than one interface

Thank you for your attention!

EXPLORE DESIGN PERFECTION