Interfaces – Contd.

CS 171: Intro to Computer Science II

Reminder: A1 due tonight 2/15 (11:59PM on Gradescope)

- Read the Canvas announcement carefully!

A1 due tom Wed 2/15 (Submission Tips!) + Quiz#4 this Fri 2/17

14 Feb at 22:15

Nosayba El-Sayed

All sections

Important reminders:

1) Our weekly Friday quiz will be open as usual this Friday 2/17 between 8AM-8PM. Review lectures 7 to 9 and their code examples (i.e. Monday 2/13's lecture & abstract class example is included).

2) A1 is due tomorrow, Wed 2/15 (at 11:59PM sharp). You'll need to submit all 4 required Java files. Submit early to Gradescope and make sure your code runs successfully (and passes our "visible" test cases--but remember that some tests are hidden!). Here's the **short video tutorial again that walks you through the submission process** .

TIPS If you get an error that Gradescope failed to execute your submission, here are some common issues I observed during this week's office hours:

- Make sure your **method signatures (headers)** match EXACTLY what is expected of you; read the starter-code comments & the handout very carefully. Some students had the wrong number, type, and/or order of expected parameters.
- Make sure your class **constructors** and parameters (if any) again match EXACTLY what is required.
- For equals() method, we have plenty of examples in our class exercises (e.g. in SalariedEmployee). Go over them and review the proper way of overriding this method in your classes (if required).
- The names of all required methods are **case sensitive**. Any typo or discrepancy with what is expected will cause gradescope to crash.

Hope this helps! And finally, go over the submission checklist at the end of A1's handout, to avoid losing points.

Abstract Class vs. Interface

- Both can't be instantiated
```
… = new AbstractCar();   //wrong
… = new CarInterface(); //wrong
```

- Abstract Classes can have constructors, instance variables, & concrete methods (i.e. non-abstract)

- Abstract Classes are *extended* by a subclass which **may** implement the abstract methods, but doesn't have to.

(if a subclass doesn't implement all abstract methods from the superclass, it must declare itself as an abstract class too…)

- Interfaces are *implemented* by another class which **must** implement its abstract methods.

Why interfaces?

- A class can only **extend** one other class, but it can **implement** *multiple* interfaces
  - This lets the class fill multiple “roles”
  - Example:

```
class MyApp extends App
        implements ActionListener, KeyListener {
   ...
} // Now you must implement ALL methods in both ActionListener and KeyListener!
```

Interface: Let's Code!

SP23_CS_171_1 › Files › CodeExamples › lectures9-10-abstraction › Interfaces

Photograph.java

Sellable.java

Transportable.java

Multiple inheritance for interfaces

Nice textbook discussion in sec 2.3.2

2.3.2 Multiple Inheritance for Interfaces

The ability of extending from more than one type is known as ***multiple inheritance***. In Java, multiple inheritance is allowed for interfaces but not for classes. The reason for this rule is that interfaces do not define fields or method bodies, yet classes typically do. Thus, if Java were to allow multiple inheritance for classes, there could be a confusion if a class tried to extend from two classes that contained fields with the same name or methods with the same signatures. Since there is no such confusion for interfaces, and there are times when multiple inheritance of interfaces is useful, Java allows interfaces to use multiple inheritance.

Btw, remember **instanceof**?

- **instanceof** returns true if a variable "is a" member of a class or *interface*!
- Example:

```
class Dog extends Animal implements Pet {...}

Dog fido = new Dog();
```

which of the following is true?

```
fido instanceof Dog

fido instanceof Animal      // superclass!

fido instanceof Pet         // interface!
```

(Answer: ALL of them are true!)

pollev.com/elsayedcs171

Question

A class can implement more than one interface.

✓ A. True

B. False

pollev.com/elsayedcs171

Question

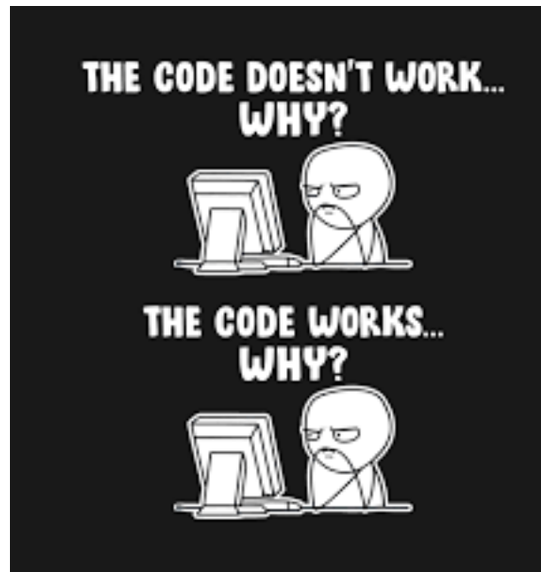
A class **ClassA** can extend an interface **ClassB** by declaring that, and by implementing any abstract methods in **ClassB**.

A. True

✓ B. False

Good to Know: `default`

- **Scenario**: You write a super cool **interface**. You provide your interface to a bunch of friends and they implement your interface with various super cool **classes** and deploy their code on their super cool websites.
- Ooops! You realize you forgot to add one of your desired methods to your interface.
- You edit your interface to add this abstract method. Now all of your friends' super cool classes won't compile because they haven't implemented this new method, and their websites are unusable.
- **In conclusion:** You are a terrible friend because you broke everyone's code ☹



Good to Know: `default`

- Solution: `default`
- The default keyword can be added to a method in an interface, allowing you to make it a concrete method with a provided default implementation
- Now your friends can either:
  - Do nothing and their class will inherit the default implementation
  - Override the default implementation to do something more specific to their class
- But no crashing code!

Good to Know: `default`

```
interface TestInterface{
    // abstract method
    public void square(int a);

    // default method
    default void show()
    {
      System.out.println("Default
          Method Executed");
    }
}
```

```
class TestClass implements TestInterface{
 // implementation of square abstract
 // method
    public void square(int a)
    {
        System.out.println(a*a);
    }

        public static void main(String
args[]) {
            TestClass d = new TestClass();
            d.square(4);
            // default method executed
            d.show();
    }
}
```

Review of some OOP concepts

- A class is like a definition of a type in Java.
- An instance of an object of a given class can be created using a constructor and the `new` operator.
- Instances are accessed through reference variables.
- A subclass can extend a superclass and use its methods and instance variables through inheritance and polymorphism.
- Abstract classes contain at least one abstract method and must be extended by a subclass that defines the abstract methods.
- Interfaces are 100% abstraction and provide a template for another class to implement all of the methods defined in the interface.

Let’s talk more about Exceptions

- Unhandled exceptions cause the code to terminate
- Reminder: how to handle Exceptions

```
try{
    // block of code that may throw exception
}
catch(Exception e){
   // deal with exception here
}
```

Exceptions Hierarchy

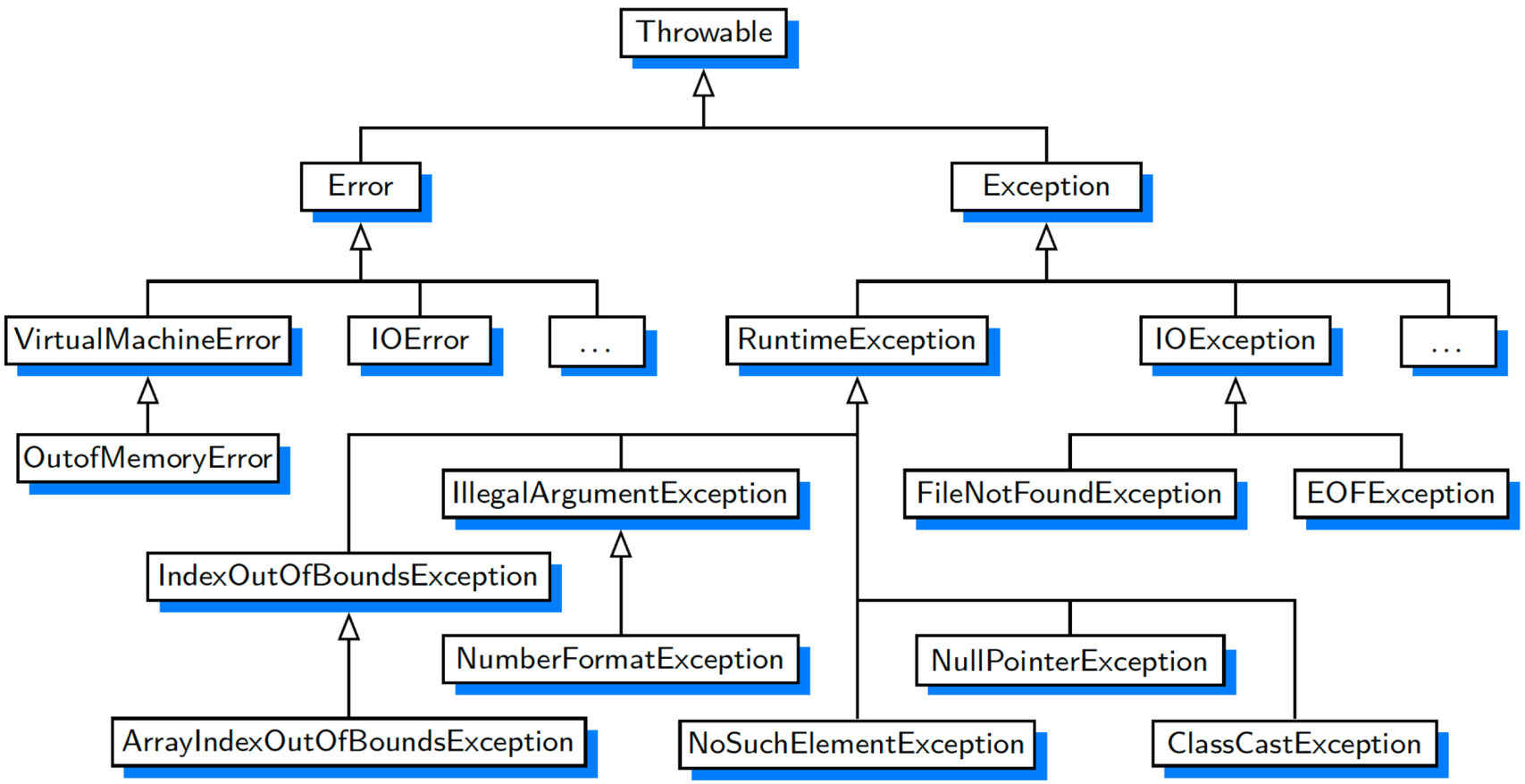


Figure 2.7: A small portion of Java's hierarchy of Throwable types.

Example: IndexOutOfBoundsException

java.lang

Class IndexOutOfBoundsException

java.lang.Object
    java.lang.Throwable
        java.lang.Exception
            java.lang.RuntimeException
                java.lang.IndexOutOfBoundsException

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

ArrayIndexOutOfBoundsException, StringIndexOutOfBoundsException

---

Exceptions & Polymorphism

- Can catch a more general exception
- “Catch-all:” use Exception itself

```
try {
    System.out.println(myNumbers[10]); // throws ArrayIndexOutOfBoundsException!
}
catch(ArrayIndexOutOfBoundsException e){
    System.out.println("Error! Bad index: " + e.getMessage());
}
catch(Exception e){
    System.out.println("Some other error occurred: " + e.getMessage());
}
```