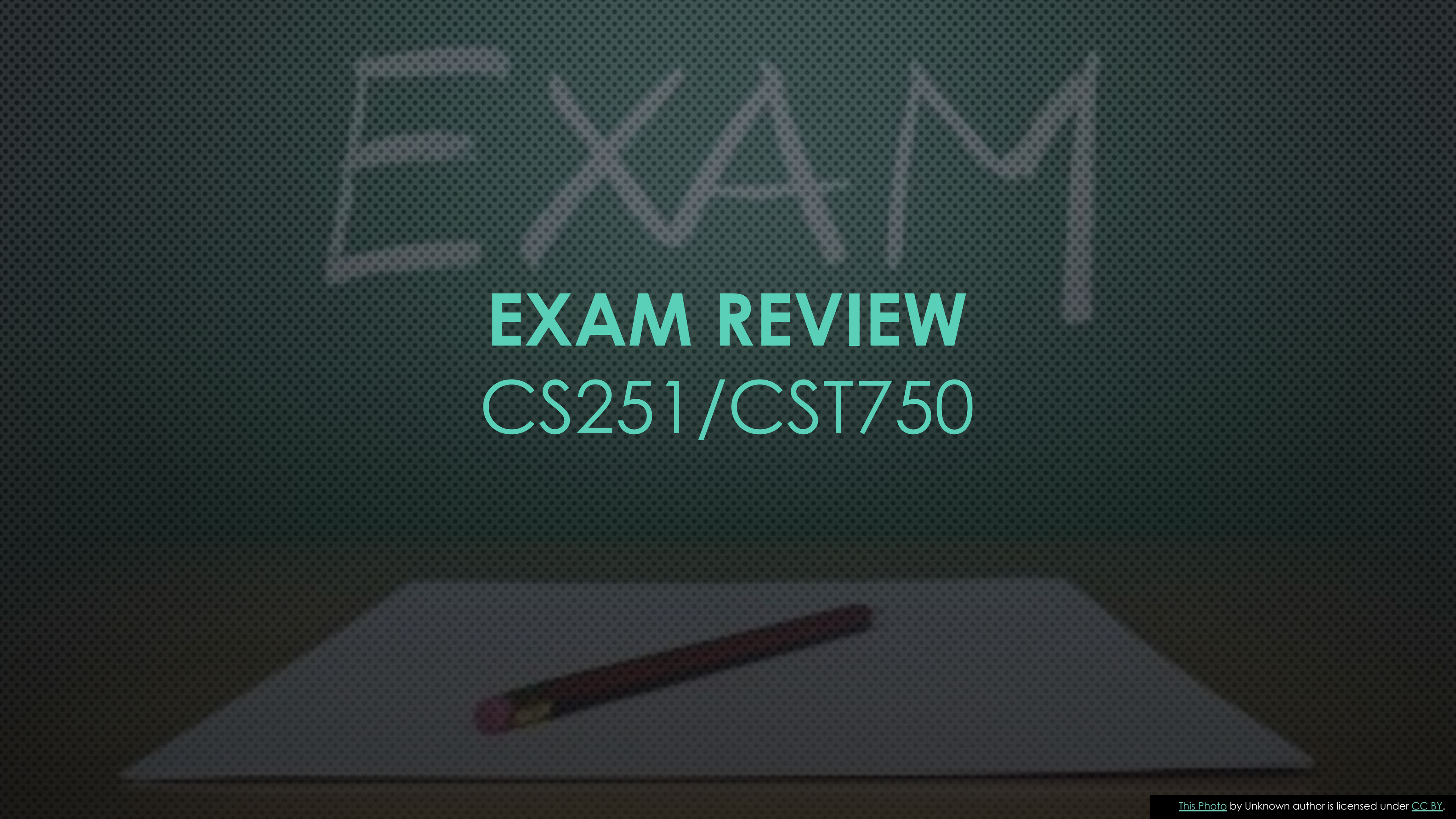


EXAM REVIEW

CS251/CST750

LET'S START WITH SOME EXAM TAKING TIPS!

Exam taking strategies

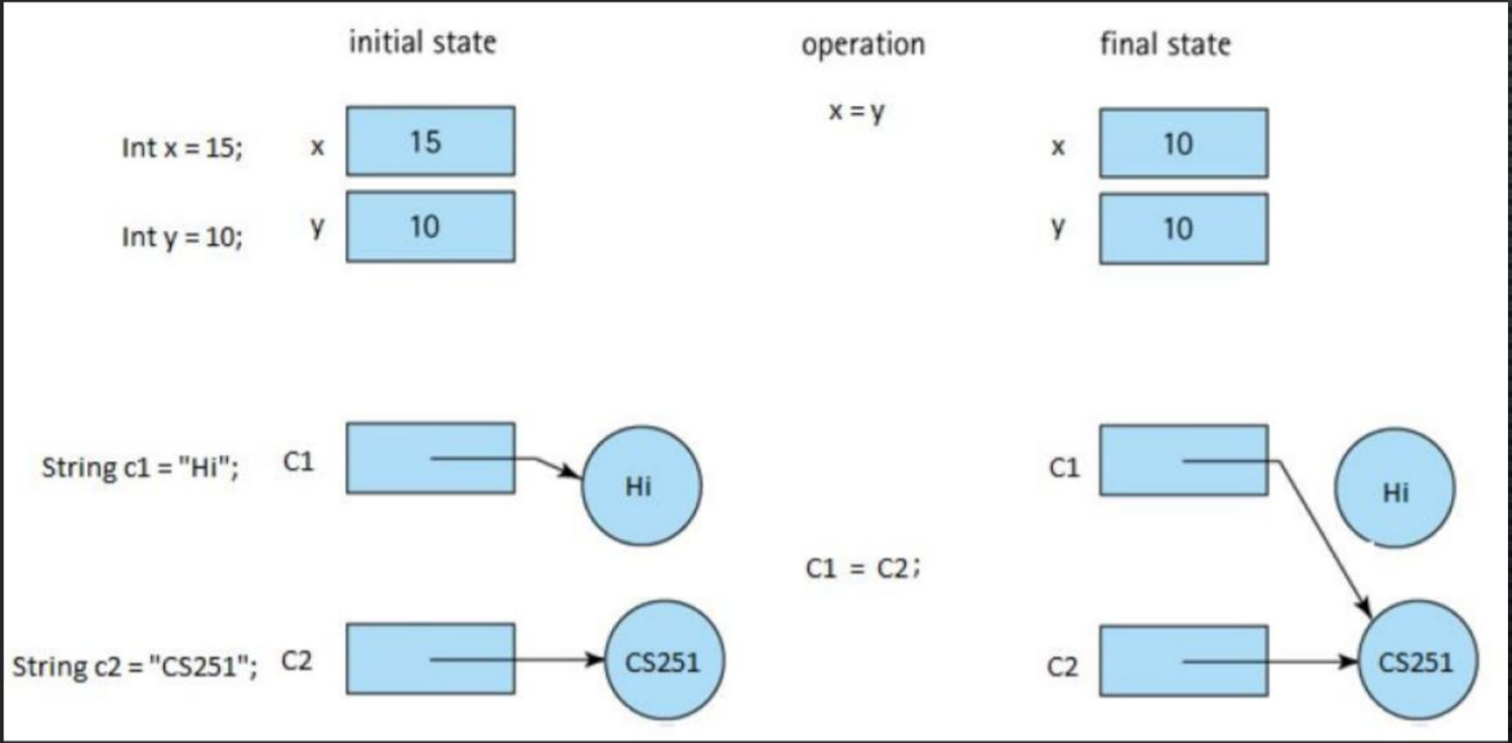
- WRITE YOUR NAME AND LAB SECTION ON YOUR EXAM.
- GLANCE OVER THE WHOLE EXAM.
- OBSERVE THE POINT VALUE OF THE QUESTIONS.
- PAY ATTENTION TO THE INSTRUCTIONS AND TO POSIBLE LAST MINUTE MODIFICATIONS ANNOUNCED BY THE INSTRUCTOR.

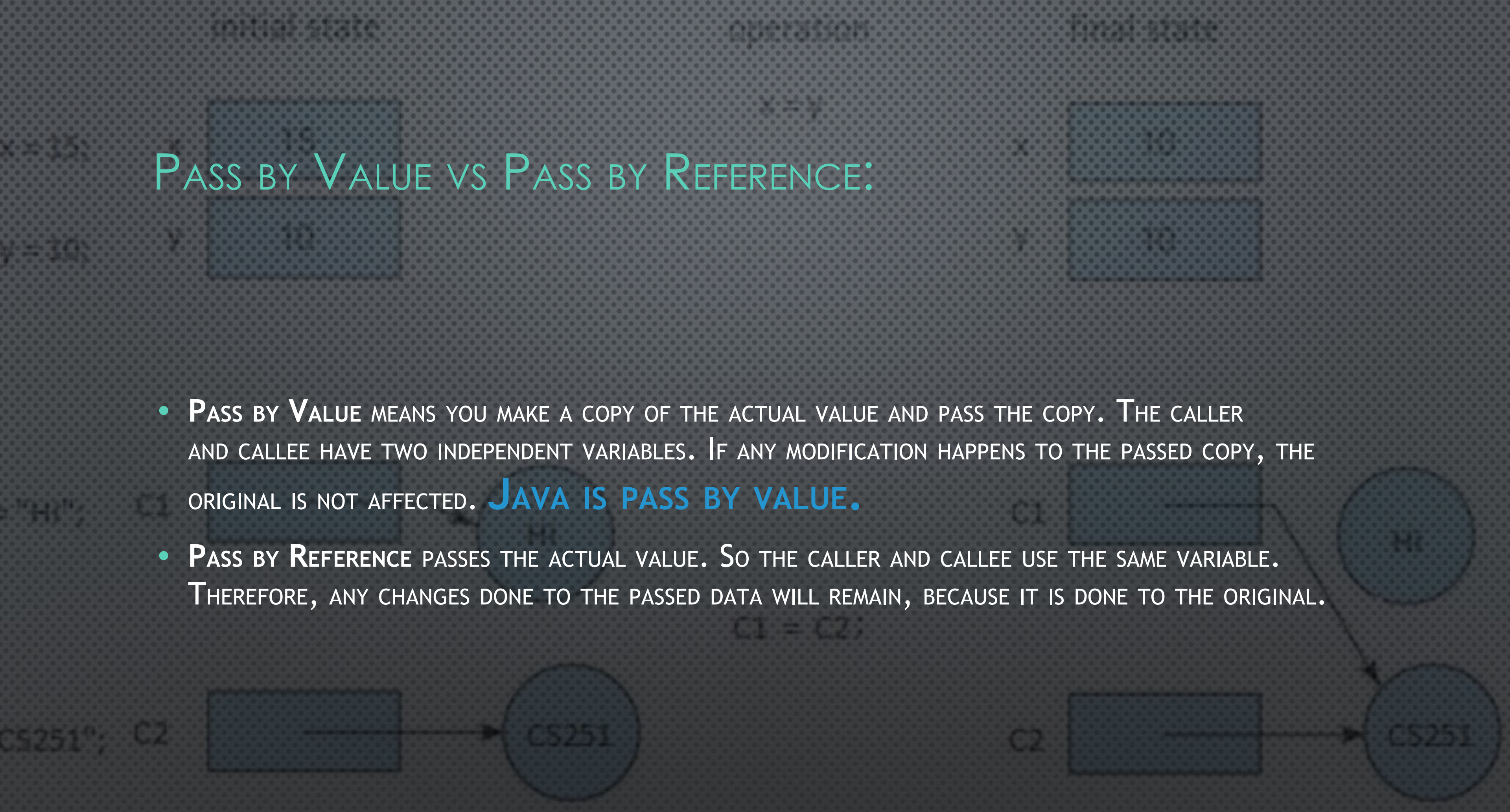
Exam taking strategies

- Be sure to identify the questions clearly.
- Tackle the questions in a sensible order. Go for the hard ones first then the easy ones or vice versa, but do not spend too much time on one question. If you are stuck, move on. You can come back to that one later.
- ASK the instructor if you do not clearly understand the directions.
- When you approach a question, UNDERLINE ALL SIGNIFICANT WORDS IN THE DIRECTIONS.
- Make sure you tried every page and every question in the exam before you hand it in.

LET'S DO SOME REVIEWING!

Data Types: Primitive vs Reference





Pass by Value vs Pass by Reference:

- **Pass by Value** means you make a copy of the actual value and pass the copy. The caller and callee have two independent variables. If any modification happens to the passed copy, the original is not affected. Java is pass by value.
- **Pass by Reference** passes the actual value. So the caller and callee use the same variable. Therefore, any changes done to the passed data will remain, because it is done to the original.

Syntax shorthands:

- **If statement:**

```
variable = (condition) ? expressionTrue : expressionFalse;
```

Ex:

```
int time = 20;
String result = (time < 18) ? "Good day." : "Good evening.";
System.out.println(result);
```

```
Good evening.
```

Syntax shorthands:

- FOR EACH (Enhanced for loop):

```
public class EnhancedForLoop
{
    public static void main(String args[])
    {
        String fruits[] = {"Apple", "Pear", "Mango"};
        //enhanced for loop
        for (String fruit:fruits)
        {
            System.out.println(fruit);
        }
    }
}
```

```
Apple
Pear
Mango
```

What is Object Oriented Programming?

- Object Oriented Programming (OOP) is a programming paradigm where a complete software operates as a bunch of objects interacting with each other. This resembles the real world.

What is an Object in Java?

- An object is a software bundle of related characteristics (fields/attributes/instance variables) and behavior (methods).
- Software objects resemble the real-world objects that we find in everyday life.
- An object is an instance of a class.

What is a class in Java?

- Classes are the blueprints for objects.
- A class describes how objects can be built.
- A class includes fields(variables), a constructor, and methods (behavior)
- Classes are a compile-time concept (means no runtime effect)

What is an instance?

- An instance of a class is a runtime-concept.
- You can have zero or more instances of a class.
- Each instance has its own copy of its instance variables (private data), but they all share the same methods (behavior)

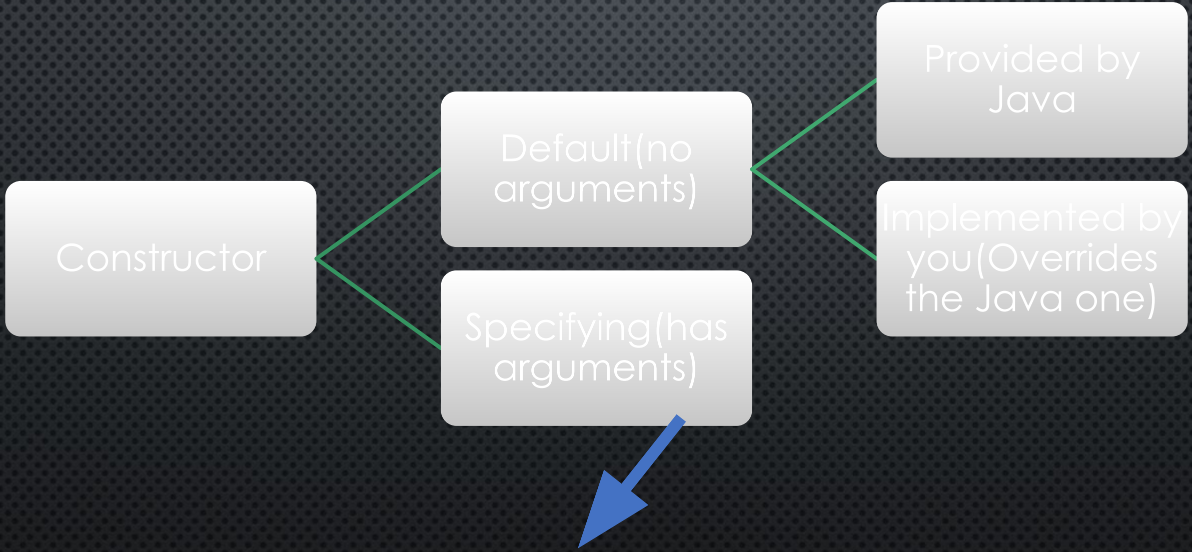
What special method allows as to instantiate/create objects out a class?

Constructor

- What must be the name of a constructor

Same as the class name

TYPES OF CONSTRUCTORS:



Constructor
Default(no arguments)
Specifying(has arguments)
Provided by Java
Implemented by you(Overrides the Java one)

This ones also invalidate the Java one

What is the signature of a method?

1. Name + parameters
2. The number of parameters in a method is part of that method's signature.
3. The types of the parameters in a method are also part of that method's signature.
4. The order of the parameters in a method is also part of that method's signature (in the case of multiple methods having the same type and number of parameters).
5. The return type of a method is **NOT** part of that method's signature.

What is method overloading?

- It means multiple methods have the same name but different parameters.
- Method overloading is also completely different than method overriding

Method overloading example

Method name

```
// Original method :
        public void show(String message){
            System.out.println(message);
        }

// Overloaded method : number of parameters is different
        public void show(String message, boolean show){
            System.out.println(message);
        }
// Overloaded method : type of parameter is different
        public void show(Integer message){
            System.out.println(message);
        }
// Not a Overloaded method : only return type is different
        public boolean show(String message){
            System.out.println(message);
            return false;
        }
```

OOP Principles:

- **Encapsulation:** Hiding the internals of a class
  - Objects maintain their own state.
- **Abstraction:** Using a class without knowing its internals
  - Objects provide methods that can be used without knowing the details of the class.
- **Inheritance:** A class sharing the internals and behavior of another class
  - Objects can reuse common logic shared with other classes.
- **Polymorphism:** Sharing a method without having the same internal implementation
  - Objects can implement common logic differently.

What are some of the testing methods we have seen in this course:

- assertEquals( )
- assertTrue( )
- assertFalse( )
- assertNull( )
- assertArrayEquals( )

Testing: The setUp() method

```
import static org.junit.Assert.*;
import junit.framework.TestCase;

public class TestReview extends TestCase {

    public void setUp() {
        int[] d = {3,4};
        //all the code added in this method will be shared by every test method
        //this method will be called before each test execution
    }

    public void test01() {
        int[] a = {1,2};
        assertEquals(a,d);
    }
}
```

Always extend this class. It turns your class into a testing class so that it can be run as a JUnit test

Testing Advice:

- It is not practical to try to test for every single input, therefore try:
  - General cases - test typical operation
  - Sanity checks - test basic assumptions
  - Edge/corner cases - test extreme values
  - Special cases - test unusual values

Debugging in Eclipse

- Eclipse has a debugger which is a tool that we can use to find problems in our code. It does two main things:
    1. It lets you freeze and advance execution
    2. It gives you information about the current state of your program

Debugging in Eclipse: Steps

- Add a breakpoint
- Run your program in debug mode
- Switch into debugging perspective
- Debug

Debugging: keep in mind!

- You will never go back!!
- The debugging run is a controlled regular run, so it always goes forward. If you need to step back, stop and rerun de debugger.
- If you get lost and start seeing JUnit code or "source not found" or class docs or you just completely lost touch with your code, most likely you went too far, so stop, modify your breakpoints if needed, and rerun the debugger again.

Debugging: Advancing

- While debugging you should pay attention to 3 important things:
  - The stack trace: shows an ordered list of events that the run will go through, the trace of method calls and the lines where things are happening.
  - The navigation tools: step into, step over, step return, resume.
  - The variables view: you can keep track and follow what happens to your data as you progress in the run.

Debugging: Navigation tools

- Step Over (F6) Execute until the current line is finished then freeze
- Step Into (F5) Execute until a different method is called, then freeze inside that method (or just finish the line if no method call is on this line)
- Step Return (F7) Execute until the current method returns, then freeze where this method was called
- Resume (F8) Execute until another breakpoint is found

What is method overriding?

- Declaring a method in a **sub class** which is already present in its **parent class** is known as method overriding.

Why?

- Overriding is done so that a child class can give its own implementation to a method which is already provided by the parent class.
- In this case the method in parent class is called overridden method and the method in child class is called overriding method.

What are some of the inherited methods from the class Object that we have used in this course?

- Equals()
- ToString()
- GetClass()

Method overriding example

```
class Human{
      //Overridden method
      public void eat()
      {
         System.out.println("Human is eating");
      }
 }
class Boy extends Human{
      //Overriding method
      @Override
      public void eat(){
         System.out.println("Boy is eating");
      }
}
```

```
public static void main( String args[]) {
         Boy b = new Boy();
         Human h = new Human();
         Human h2 = new Boy();

         b.eat();
         h.eat();
         h2.eat();
      }

// What will the output be?
```

Boy is eating
Human is eating
Boy is eating (this also shows polimorphysm)

Key things about overriding methods

- You can only override methods in a sub class. You cannot override methods in the same class.
- The signature (name + arguments) of a method must be the same for the Super class and its Sub classes, and for an interface and its implementations.
- The overriding method cannot reduce accessibility of the overridden method. For example if the overridden method is public then the overriding method cannot be protected or private; however, the opposite is true. The overriding method can increase in accessibility.
- You cannot override a private, static or final method.
- If you are extending an abstract class or implementing an interface, you will need to override all abstract methods.
- Always use @Override to mark the overriding method in Java. Although this is not required by the compiler, it is good coding practice to do so.

What is an abstract class in Java?

- An abstract class is a class considered as non-fully implemented, because although it has fully implemented methods, it might contain non-implemented methods (abstract methods).
- We cannot instantiate an abstract class in Java. In order to use it, we must extend it and implement all its abstract methods.
- To declare an abstract class we use the keyword abstract.
- in Java, a class can be abstract without specifying any abstract method.
- Also, having at least one abstract method in a class will turn that class into an abstract class and Eclipse will demand us to declare it as such.

What is an interface in Java?

- An interface is a "**completely abstract class**" that is used to group related methods which are all abstract methods (they do not have bodies).
- Like **abstract classes**, interfaces **cannot** be used to create objects
- To use the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the `implements` keyword (instead of `extends`) and all those methods must be overriden.
- Interface methods are by default `abstract` and `public`
- Interface attributes are by default `public`, `static` and `final`
- An interface cannot contain a constructor

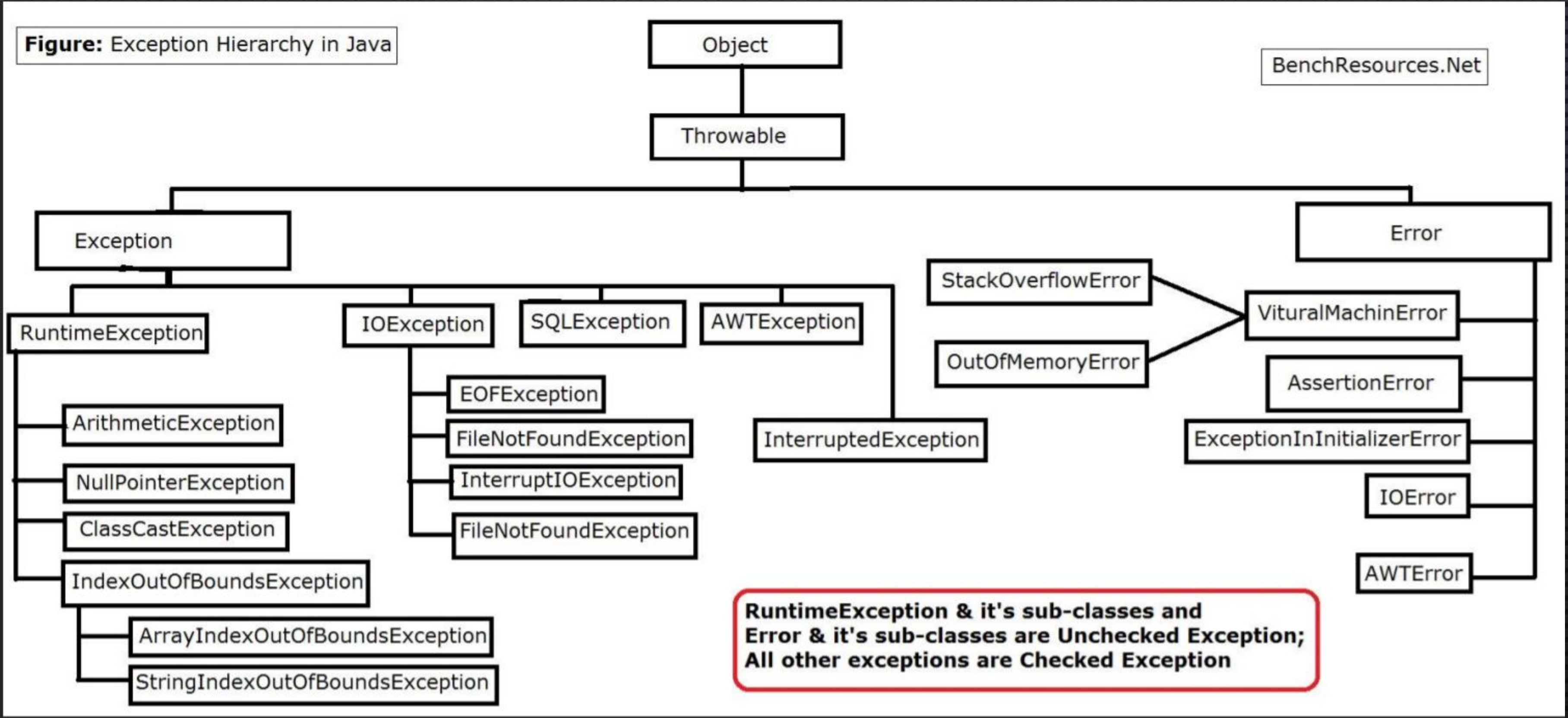
Some of the main differences between abstract classes and interfaces in Java:

- abstract classes can contain non-abstract methods, but interfaces cannot (Except: default and static methods, not the scope of this course).
- Abstract classes can have constructors, but interfaces cannot.
- Keyword extends for abstract classes and implements for interfaces

Association, Aggregation, and Composition in OOP?

- When an object is related to another with a has-a relationship it is called association. It has two forms: aggregation and composition.
- Let class A and class B, if a = new A(), b = new B() and a has a b then:
- Aggregation is the weakest form of association where if a is destroyed b survives
- Composition is the strongest form of association where if a is destroyed, b is also gone.
- For example, a city is an aggregation of people, and people is a composition of body parts.

Exception handling



RuntimeException & it's sub-classes and Error & it's sub-classes are Unchecked Exception; All other exceptions are Checked Exception

How do we deal with exceptions?

THROW AN EXCEPTION

or

TRY TO CATCH IT

Throw and Throws keywords

- Throw
  - The throw keyword in Java is used to explicitly throw an exception from a method or any block of code. We can throw either checked or unchecked exception. The throw keyword is mainly used to throw custom exceptions.

Ex: **throw new** NullPointerException("null found");

Throw and Throws keywords

- Throws
  - This keyword is used in the signature of a method to indicate that this method might throw exceptions. The caller to these methods has to handle the exceptions using a try-catch block.

Syntax:

```
type method_name(parameters) throws exception_list
exception_list is a comma separated list of all the
exceptions which a method might throw.
```

```
Ex: public int div(int d) throws ArithmeticException {
        return d / 0;
    }
```

Try Catch Finally

- **Control flow in try-catch clause OR try-catch-finally clause**
  - **Case 1:** Exception occurs in try block and handled in catch block
  - **Case 2:** Exception occurs in try-block is not handled in catch block
  - **Case 3:** Exception doesn't occur in try-block

Try-catch and Try-catch-finally

- **CASE 1: Exception occurs in try block and handled in catch block**
  - If a statement in try block raised an exception, then the rest of the try block does not execute, and control passes to the **corresponding** catch block. After executing the catch block, the control will be transferred to finally block(if present) and then the rest of the program will be executed.

Try-catch and Try-catch-finally

- **CASE 2: Exception occurred in try-block is not handled in catch block**
  - In this case, default handling mechanism is followed. If finally block is present, it will be executed followed by default handling mechanism.

Try-catch and Try-catch-finally

- **CASE 3: Exception doesn't occur in try-block**
  - In this case catch block never runs as they are only meant to be run when an exception occurs. finally block(if present) will be executed followed by rest of the program.

Remember: Exceptions are Objects in Java

- So, we can create our own exceptions
- Our custom exceptions must extend Throwable or any of its subclasses

Important keywords in Java:

- Class
- Final
- Static
- Abstract
- Interface
- Implements
- Extends
- Public
- Private
- Protected
- Instanceof
- Super
- This
- Void
- Try
- Catch
- Finally
- Throw
- Throws
- New

NOW LET'S PLAY WITH SOME FUN JAVA QUESTIONS!

The default value of a static integer variable of a class in Java is

- (a) 0
- (b) 1
- (c) Garbage value
- (d) Null
- (e) -1.

(A) 0

What will be printed as the output of the following program?

```
public class Question
        {
            public static void main(String args[])
            {
                int i = 0;
                i = i++ + i;
                System.out.println(“I = ” +i);
            }
        }
```

(B) I = 1

- (A) I = 0
- (B) I = 1
- (C) I = 2
- (D) I = 3
- (E) Compile-time error.

Multiple inheritance means

- (A) one class inheriting from more super classes
- (B) more classes inheriting from one super class
- (C) more classes inheriting from more super classes
- (D) None of the above
- (E) (A) and (B) above.

(A)

Which statement is not true in java language?

- (A) A public member of a class can be accessed in all the packages.
- (B) A private member of a class cannot be accessed by the methods of the same class.
- (C) A private member of a class cannot be accessed from its derived class.
- (D) A protected member of a class can be accessed from its derived class.
- (E) None of the above.

(B)

To prevent any method from overriding, we declare the method as

- (A) void
- (B) const
- (C) final
- (D) abstract
- (E) None of the above

(C)

Which one of the following is not true?

- (a) A class containing abstract methods is called an abstract class.
- (b) Abstract methods should be implemented in the derived class.
- (c) An abstract class cannot have non-abstract methods.
- (d) A class must be qualified as ‘abstract’ class, if it contains one abstract method.
- (e) None of the above.

(C)

The fields in an interface are implicitly specified as

- (a) static only
- (b) protected
- (c) private
- (d) both static and final
- (e) none of the above.

(D)

What is the output of the following program:

```
public class Review {

    public static void main(String[] args) {
        int i = 1;
        System.out.print(i + " , ");
        m(i);
        System.out.println(i);
    }
    public static void m(int i) {
        i += 2;
    }
}
```

(C) 1 , 1

- (A) 1 , 3
- (B) 3 , 1
- (C) 1 , 1
- (D) 1 , 0
- (E) None of the above.

How about now:

```
public class Review {
    public static int i = 1;

    public static void main(String[] args) {

        System.out.print(i + " , ");
        m(i);
        System.out.println(i);
    }

    public static void m(int i) {
        i += 2;
    }
}
```

(C) 1 , 1

- (A) 1 , 3
- (B) 3 , 1
- (C) 1 , 1
- (D) 1 , 0
- (E) NONE OF THE ABOVE.

How about now:

```
public class Review {
    public static int i = 1;
    public static void main(String[] args) {

        System.out.print(i + " , ");
        i = m(i);
        System.out.println(i);
    }
    public static int m(int i) {
        return i += 2;
    }
}
```

(A) 1 , 3

- (A) 1 , 3
- (B) 3 , 1
- (C) 1 , 1
- (D) 1 , 0
- (E) None of the above.

How about now:

```
public class Review {
    public int i = 1;
    public static void main(String[] args) {
        System.out.print(i + " , ");
        i = m(i);
        System.out.println(i);
    }
    public static int m(int i) {
        return i += 2;
    }
}
```

(E) Error at line 4

- (A) 1 , 3
- (B) 3 , 1
- (C) 1 , 1
- (D) 1 , 0
- (E) None of the above.

How about now:

```
public class Review {
    public int i = 1;
    public static void main(String[] args) {
        Review r = new Review();
        System.out.print(r.i + " , ");
        m(r.i);
        System.out.println(r.i);
    }
    public static void m(int i) {
        i += 2;
    }
}
```

(C) 1 , 1

- (A) 1 , 3
- (B) 3 , 1
- (C) 1 , 1
- (D) 1 , 0
- (E) None of the above.

How about now:

```
public class Review {
    public int i = 1;
    public static void main(String[] args) {
        Review r = new Review();
        System.out.print(r.i + " , ");
        r.m(r.i);
        System.out.println(r.i);
    }
    public void m(int i) {
        i += 2;
    }
}
```

(C) 1 , 1

- (A) 1 , 3
- (B) 3 , 1
- (C) 1 , 1
- (D) 1 , 0
- (E) None of the above.

How about now:

```
public class Review {
    public int i = 1;
    public static void main(String[] args) {
        Review r = new Review();
        System.out.print(r.i + " , ");
        r.m(r.i);
        System.out.println(r.i);
    }
    public void m(int i) {
        this.i += 2;
    }
}
```

(A) 1 , 3

- (A) 1 , 3
- (B) 3 , 1
- (C) 1 , 1
- (D) 1 , 0
- (E) None of the above.

How about now:

```
public class Review {
    public static int i = 1;
    public static void main(String[] args) {
        Review r = new Review();
        System.out.print(i + " , ");
        i = r.m(i);
        System.out.println(i);
    }
    public int m(int i) {
        return i += 2;
    }
}
```

(A) 1 , 3

- (A) 1 , 3
- (B) 3 , 1
- (C) 1 , 1
- (D) 1 , 0
- (E) None of the above.

What about now:

```
public class Review {
    public static final int i = 1;
    public static void main(String[] args) {
        Review r = new Review();
        System.out.print(i + " , ");
        i = r.m(i);
        System.out.println(i);
    }
    public int m(int i) {
        return i += 2;
    }
}
```

(E) Error at line 6

- (A) 1 , 3
- (B) 3 , 1
- (C) 1 , 1
- (D) 1 , 0
- (E) None of the above.

Which of the following is not true?

- (a) An interface can extend another interface.
- (b) A class which is implementing an interface must implement all the methods of the interface.
- (c) An interface can implement another interface.
- (d) An interface is a solution for multiple inheritance in java.
- (e) None of the above.

(C)

Which of the following is true?

- (a) A finally block is executed before the catch block but after the try block.
- (b) A finally block is executed, only after the catch block is executed.
- (c) A finally block is executed whether an exception is thrown or not.
- (d) A finally block is executed, only if an exception occurs.
- (e) None of the above.

(C)

Among these expressions, which is(are) of type String?

- (A) "0"
- (B) "AB" + "CD"
- (C) '0'
- (D) Both (A) and (B)
- (E) (A), (B) and (C)

(D)

Consider the following code fragment

```
Rectangle r1 = new Rectangle();
r1.setColor("blue");
Rectangle r2 = r1;
r2.setColor("red");
```

After the above piece of code is executed, what are the colors of r1 and
r2 (in this order)?

(C)

- (A) blue and red
- (B) blue and blue
- (C) red and red
- (D) red and blue
- (E) None of the above.

What is the type and value of the following expression?

-4 + 1/2 + 2*-3 + 5.0

- (A) int -5
- (B) double -4.5
- (C) int -4
- (D) double -5.0
- (E) None of the above.

(D) ½ results in 0 not 0.5

What is printed by the following statement?

```
System.out.print("Hello,\nworld!");
```

(C)

- (A) Hello, \nworld!
- (B) Hello, world!
- (C) Hello,

  world!
- (D) "Hello, \nworld!"
- (E) None of the above.

Consider these two methods (within the same class)

(D)

```
public static int foo(int a, String s) {
    s = "Yellow";
    a = a + 2;
    return a;
}
public static void bar() {
    int a = 3;
    String s = "Blue";
    a = foo(a,s);
    System.out.println("a="+a+" s="+s);
}
public static void main(String[] args) {
    bar();
}
```

What is printed on execution of these methods?

- (A) a = 3 s = Blue
- (B) a = 5 s = Yellow
- (C) a = 3 s = Yellow
- (D) a = 5 s = Blue
- (E) None of the above.

Consider the following class definition:

```
public class MyClass
{
    private int value;
    public void setValue(int i){ / code / }
    // Other methods…
}
```

The method setValue assigns the value of i to the instance field value. What could you write for the implementation of setValue?

(D)

- (a) value = i;
- (b) this.value = i;
- (c) value == i;
- (d) Both (A) and (B)
- (e) (A), (B) and (C)

Which of the following is TRUE?

- (A) In Java, an instance field declared public generates a compilation error.
- (B) int is the name of a class available in the package java.lang
- (C) Instance variable names may only contain letters and digits.
- (D) A class has always a constructor (possibly automatically supplied by the Java compiler).
- (E) The more comments in a program, the faster the program runs.

(D)

A constructor

- (a) Must have the same name as the class it is declared within.
- (b) Is used to create objects.
- (c) May be declared private.
- (d) Both (A) and (B) above.
- (e) (a), (b) and (c) above.

(E)

How can we access/modify the private fields of an object?

- By setting up and using getters and setters

An overloaded method consists of

- (a) The same method name with different types of parameters
- (b) The same method name with different number of parameters
- (c) The same method name and same number and type of parameters with different return type
- (d) Both (a) and (b)
- (e) (a), (b) and (c)

(D)

A protected member can be accessed in

- (a) a subclass of the same package
- (b) a non-subclass of the same package
- (c) a non-subclass of different package
- (d) a subclass of different package
- (e) the same class.

Which is the false option?

(C)

All exception types are subclasses of the built-in class

- (a) Exception
- (b) RuntimeException
- (c) Error
- (d) Throwable
- (e) None of the above.

(D)

When an overridden method is called from within a subclass, it will always refer to the version of that method defined by the

- (a) Super class
- (b) Subclass
- (c) Compiler will choose randomly
- (d) Interpreter will choose randomly
- (e) None of the abvove.

(B)

Use the following declaration and initialization to evaluate the Java expressions

```
int a = 2, b = 3, c = 4, d = 5;
float k = 4.3f;

System.out.println( --b * a + c *d --);
```

(B)

- (A) 21
- (B) 24
- (C) 28
- (D) 26
- (E) 22.

Fill the Blank

- Members of a class specified as ……..private…………… are accessible only to methods of that class.

Fill the Blank

- In Java, a try block should immediately be followed by one or more ……………………catch blocks.

Fill the Blank

- In object-oriented programming, the process by which one object acquires the properties of another object is called inheritance ....

Fill the Blank

constructor

- In a class definition, the ………………………………… is a special method provided to be called to create an instance of that class

Consider the following statements:

I. A class can be declared as both abstract and final simultaneously.

II. A class declared as final can be extended by defining a sub-class.

III. An abstract class can be extended more than once.

IV. The class Object defined by Java is not a super class of all other classes.

Which ones are correct?

- (A) Both (I) and (II)
- (B) Both (III) and (IV)
- (C) Both (I) and (III)
- (D) Both (II) and (IV)
- (E) Only (III)

(E)

Consider the following data types in Java :

A. int

B. boolean

C. double

D. String

E. Array.

Which of them are primitive data types?

A, B, and C

OOP quick review:

- Real-world objects contain .... attributes and .... behaviour
- A software object's attributes are stored in .... fields
- A software object's behavior is exposed through .... methods
- Hiding internal data from the outside world and accessing it only through publicly exposed methods is known as data .... encapsulation
- A blueprint for a software object is called a .... class
- Common behavior can be defined in a .... superclass and inherited into a subclass .... using the .... extends keyword.
- A collection of methods with no implementation is called an .... interface

Quick answer Questions:

- Can we overload a static method in Java?

  Yes, they will all be shared by the class objects

- Can we override a static method in Java?

  No, it is a shared behavior among all the class objects

- What are some ways that we can prevent a method from being overriden?

  Declare it as final or static or private

- How do you call the superclass version of an overriding method in a subclass?

  Use the keyword super, ex: super.method()

Quick Questions:

- Can an interface extend more than one interface in Java?

  Yes

- Can a class extend more than one class in Java?

  No

- What is the main idea (reason) behind these last two answers?

  Avoiding ambiguity

Quick Questions:

- What kind of relationship exists between classes that follow the principle of Inheritance?

  Is-a relationship

- What kind of relationship exist between classes that implement association (aggregation and composition)?

  Has-a relationship

LET'S GET INTO SOME DEEP THINKING!

Why to use interfaces?

- To achieve "multiple inheritance"
- A class can only inherit from one superclass; however, a class can **implement** multiple interfaces. Why?

Because they are all fully abstract allowing a class to inherit more than one method with the same signature. Since all interface methods are abstract, there is no room for ambiguity.

Why do we want to override toString and equals?

- We will want the string representation of our objects and their comparison criteria to match our design. For this we want to modify the behavior of these methods accordingly.
- We will also need to have these methods properly implemented because they will be used by Java in several important places like when:
  - using System.out.print()
  - using the debugger displaying values of objects and variables
  - using the assertion methods when writing our tests

Consider this code:

```
public class Phone {
   //Overridden method
   public void call()
   {
      System.out.println("Calling....");
   }

}

public interface Electronic{
   public void on();
}
```

```
public class Iphone extends Phone
implements Electronic{

      //Overriding methods
      @Override
      public void on(){
         System.out.println("iPhone is on");
      }

      @Override
      public void call() {
         System.out.println("iPhone is
calling....");
      }
}
```

Using the previous slide, tell the possible output of each method call in this Driver and spot the problems with it, if any:

```
public static void main( String args[]) {
    Iphone i = new Iphone();
    Phone p = new Phone();
    Phone p2 = new Iphone();
    Iphone i2 = new Phone();                Cannot convert from Phone to Iphone

    Electronic e = new Iphone();
    Electronic e2 = new Electronic();       Error, cannot instantiate an interface
    Electronic e3 = new Phone();            Error, there is no inheritance relationship between Phone and Electronic
    i.on();                                 iPhone is on
    i.call();                               iPhone is calling....
    p.on();                                 Error, on is undefined for Phone
    p.call();                               Calling....
    p2.on();                                Error, on is undefined for Phone
    ((Iphone)p2).on();                      iPhone is on
    ((Iphone)p).on();                       Error, ClassCastException Phone is not an Iphone

    p2.call();                              iPhone is calling....
    e.call();                               Error, call is undefined for Electronic
    e.on();                                 iPhone is on
}
```

Testing class, JUnit in Java

```
import static org.junit.Assert.*;
import junit.framework.TestCase;

public class TestReview extends TestCase {
    public void test01() {
        int[] a = {1,2};
        int[] b = {1,2};
        int[] c = a;

        assertEquals(1,1);          // this is a pass
        assertEquals(a, b);        // this do not pass, pointers are different
        assertEquals(a,c);         // this is a pass, pointers are the same
        assertArrayEquals(a,b);          // this is a pass, contents are the
same
        AssertArrayEquals(a,c);          // this is a pass, contents are the
same
    }
```

Debugging: give the proper tool to use in the following cases

- I'm debugging and I want to look inside a method that is about to be called.

  Step into

- I'm debugging and I want to look at what happens after the current line executes.

  Step over

- I'm debugging and I am not interested in the rest of the current method. I want to go back to the method that called this one.

  Step return

Debugging: give the proper tool to use in the following cases

- I'm debugging and I am not interested in the current method. I want to go to the next breakpoint that I placed.

  Resume

- I'm debugging and I am finished looking at the execution for this run. I want to let the program finish executing.

  Regular run

- I'm debugging and I accidentally stepped into a Java library method without source code. I want to go back!

  Terminate and rerun the debugger

NOW LET'S THINK EVEN DEEPER!

Why?

- Why do we use OOP?
- Why use encapsulation?
- Why inheritance?
- Why to catch an exception?
- Why do we care about polymorphism?
- Why would we need a static method?
- Why would we want to have a final variable?
- What is the difference between an instance method and a static method?

THANK YOU

By Yazan Salem

UWM 2021