**Question Title**

6. An application has four classes: **A**, **B**, **C** and **D**. Classes **B** and **C** are subclasses of **A**. In classes **A**, **B** and **C** there are different attributes and a method called **checkBalance()** with the same signature but which performs operations that return values ​​of type **double**, however, different. Class **D** is the main class. Within the **main()** method of class **D**, the following instructions were typed:  
  
**A\* obj = new B();**  
**double v = obj->checkBalance();**  
  
**A\* obj1 = new C();**  
**double v1 = obj1->checkBalance();**  
  
When these lines were executed, the variable **v** received the value 100.00 and the variable **v1** received the value 125.00. Note that both **obj** and **obj1** are type **A** objects, however, in addition to having different attributes, when calling the **checkBalance()** method through these objects, the return contained in variables **v** and **v1** was different. This shows an example of:

Encapsulation

Method overload

Multiple inheritance

Generalization

Polymorphism

**SOLUTION**

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance [1]. Polymorphims in object-oriented programming is the ability to create a variable, a function, or an object that has more than one form – Superset of overloading, overriding [2].

Like we specified in the previous chapter; Inheritance lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different tasks. This allows us to perform a single action in different ways [1].

Runtime Polymorphism (or Dynamic polymorphism) is also known as Dynamic Method Dispatch. Dynamic polymorphism is a process in which a call to an overridden method is resolved at runtime, thats why it is called runtime polymorphism. I have already discussed method overriding in detail in a separate tutorial, refer it: [Method Overriding in Java](https://beginnersbook.com/2014/01/method-overriding-in-java-with-example/) [3].

**Example**  
In this example presented in [3], we have two classes ABC and XYZ. ABC is a parent class and XYZ is a child class. The child class is overriding the method myMethod() of parent class. In this example we have child class object assigned to the parent class reference so in order to determine which method would be called, the type of the object would be determined at run-time. It is the type of object that determines which version of the method would be called (not the type of reference).

To understand the concept of overriding, you should have the basic knowledge of [inheritance in Java](https://beginnersbook.com/2013/03/inheritance-in-java/).

class ABC{

public void myMethod(){

System.out.println("Overridden Method");

}

}

public class XYZ extends ABC{

public void myMethod(){

System.out.println("Overriding Method");

}

public static void main(String args[]){

ABC obj = new XYZ();

obj.myMethod();

}

}

**Output:**

When an overridden method is called through a reference of parent class, then type of the object determines which method is to be executed. Thus, this determination is made at run time.  
Since both the classes, child class and parent class have the same method animalSound. Which version of the method (child class or parent class) will be called is determined at runtime by JVM.

**Few more overriding examples:**

ABC obj = new ABC();

obj.myMethod();

// This would call the myMethod() of parent class ABC

XYZ obj = new XYZ();

obj.myMethod();

// This would call the myMethod() of child class XYZ

ABC obj = new XYZ();

obj.myMethod();

// This would call the myMethod() of child class XYZ

In the third case the method of child class is to be executed because which method is to be executed is determined by the type of object and since the object belongs to the child class, the child class version of myMethod() is called.

**Conclusion**

Back to the original problem, the example abovementioned explains why the solution is a **polymorphism.**

**References**

[1] <https://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/>

[2] Guéhéneuc, Yann-Gael. “Encapsulation, Inheritance, Types, Overloading, Overriding, Polymorphism, and Abstraction”, École Polytechnique Montreal, CSE3009 - Software Archtecture and Design, 2013, pp. 29-55

[3] https://www.w3schools.com/cpp/cpp\_constructors.asp