Introduction to Java

LESSON 4

Learning Outcomes : Lesson 4

- Interface
- Polymorphism
- Typecasting
- Upcasting
- Downcasting
- Instanceof Keyword

Interface

- An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.
- Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.
- Interfaces have the following properties :
  - An interface is implicitly abstract. You do not need to use the **abstract** keyword while declaring an interface.
  - Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
  - Methods in an interface are implicitly public.

Interface vs Class

- An interface is similar to a class in the following ways :
  - An interface can contain any number of methods.
  - Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.
  - An interface is written in a file with a .java extension, with the name of the interface matching the name of the file.
  - The byte code of an interface appears in a **.class** file.

Interface vs Class

- An interface is different from a class in several ways including :
  - You cannot instantiate an interface.
  - An interface does not contain any constructors.
  - All of the methods in an interface are abstract.
  - An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
  - An interface is not extended by a class; it is implemented by a class.
  - An interface can extend multiple interfaces.

Implementing Interface

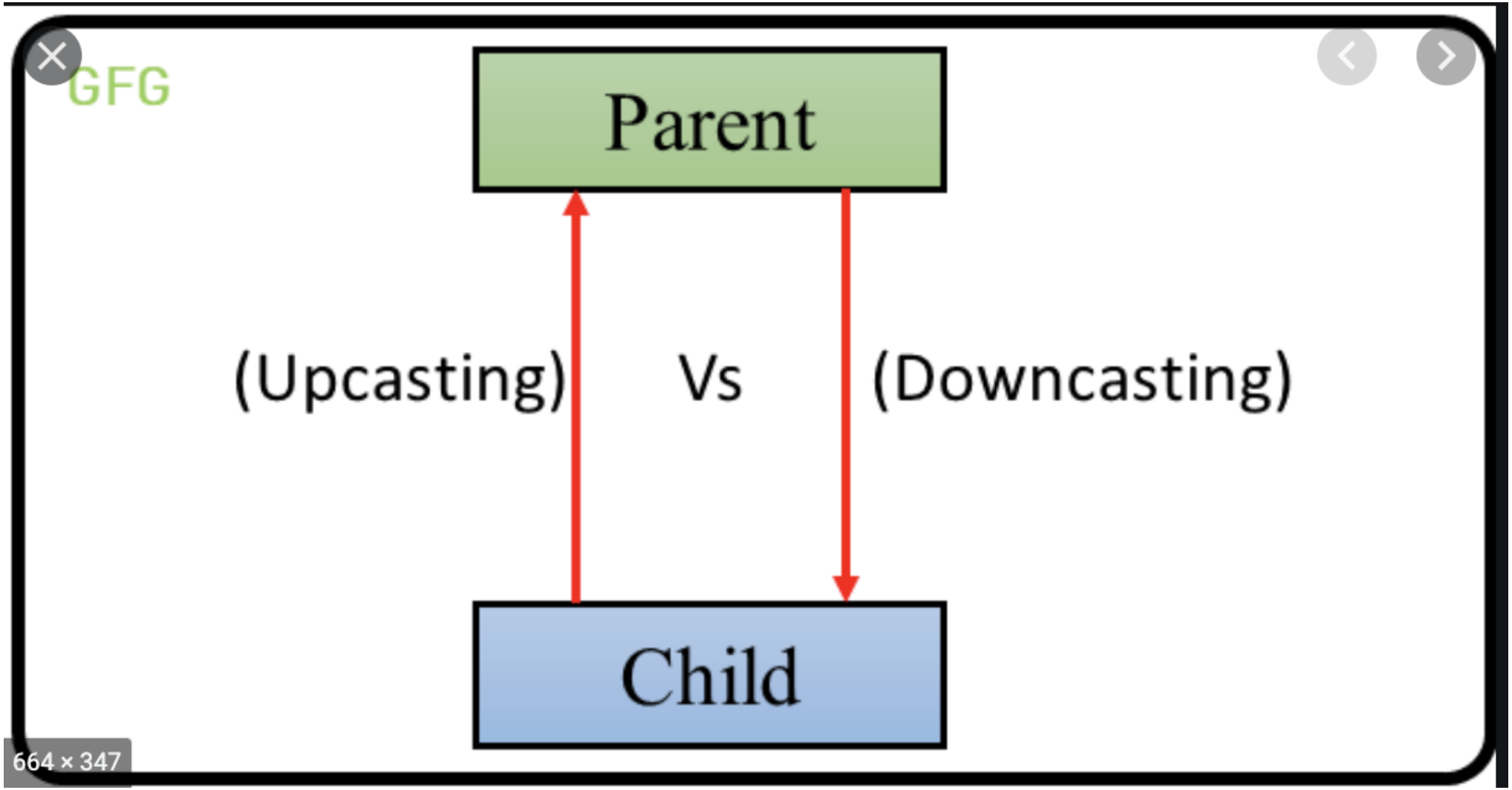
- Interfaces are used for full abstraction. Since methods in interfaces do not have body, they have to be implemented by the class before you can access them. The class that implements interface must implement all the methods of that interface. Also, java programming language does not allow you to extend more than one class, However you can implement more than one interfaces in your class.
- implements keyword is used by classes to implement an interface.
- Class that implements any interface must implement all the methods of that interface, else the class should be declared abstract.
- All the interface methods are by default abstract and public.
- Variables declared in interface are public, static and final by default.

Polymorphism

- Polymorphism is one of the OOPs feature that allows us to perform a single action in different ways.
- Polymorphism is the capability of a method to do different things based on the object that it is acting upon. In other words, polymorphism allows you define one interface and have multiple implementations.
- There are two types of polymorphism in Java:
  - compile-time polymorphism
  - runtime polymorphism.
- We can perform polymorphism in java by method overloading and method overriding.
- If you overload a static method in Java, it is the example of compile time polymorphism.
- Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

Typecasting

- Typecasting is one of the most important concepts which basically deals with the conversion of one data type to another datatype implicitly or explicitly.
- Just like the datatypes, the objects can also be typecasted. However, in objects, there are only two types of objects parent object and child object. Therefore, typecasting of objects basically mean that one type of object child or parent to another.
- There are two types of typecasting.
  - Upcasting
  - Downcasting

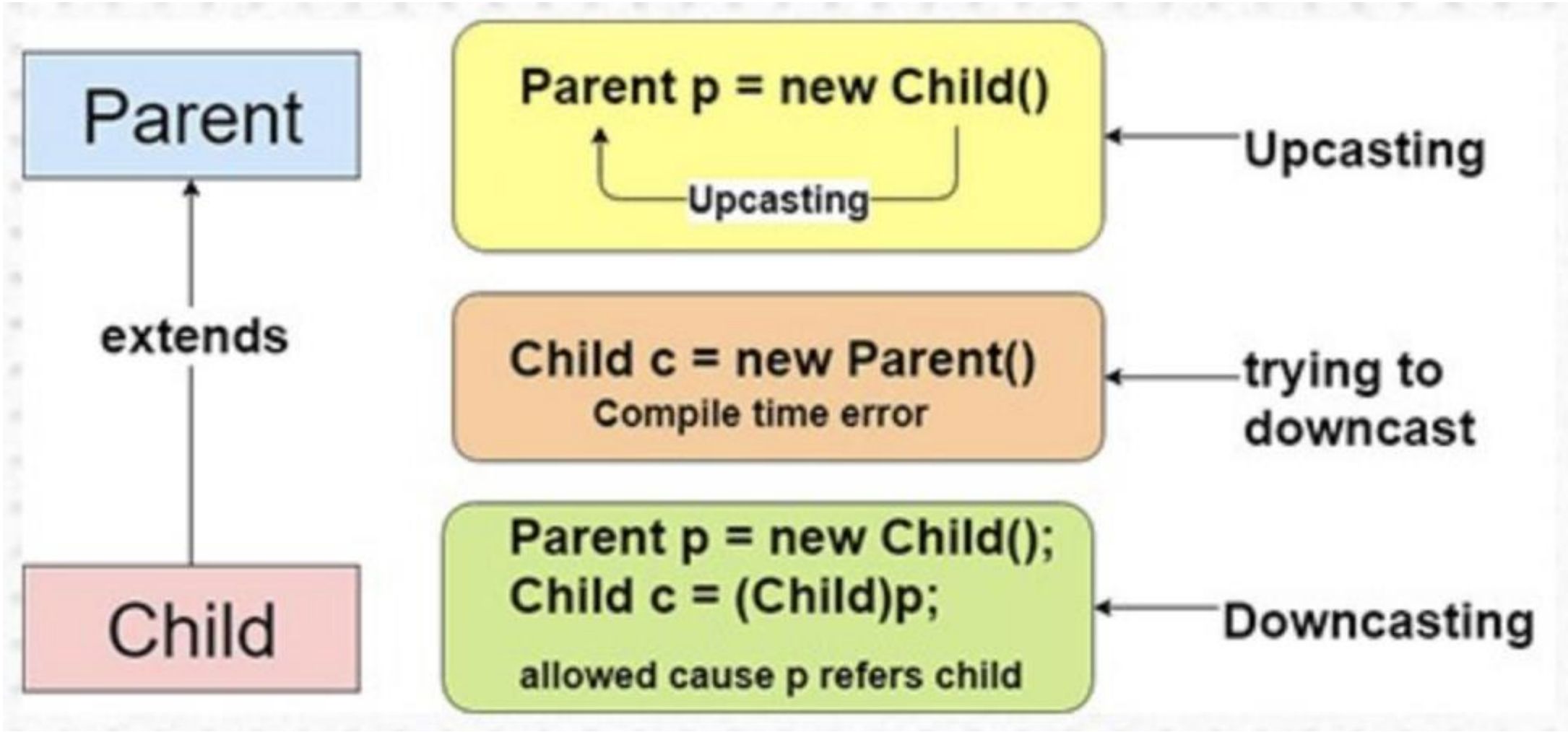

GFG
Parent
(Upcasting)
Vs
(Downcasting)
Child
664 × 347

Upcasting

- Upcasting is the typecasting of a child object to a parent object. Upcasting can be done implicitly. Upcasting gives us the flexibility to access the parent class members but it is not possible to access all the child class members using this feature. Instead of all the members, we can access some specified members of the child class.
- Parent p = new Child()
- Upcasting will be done internally and due to upcasting the object is allowed to access only parent class members and child class specified members (overridden methods, etc.) but not all members.

Downcasting

- Similarly, downcasting means the typecasting of a parent object to a child object. In other words, when sub class type is converted into super class type, it is called downcasting.
- Downcasting cannot be implicitly.
- If we do downcasting, there will not be any compiler error. But, there will be runtime exception java.lang.ClassCastException.
- Child c= (Child)new Parent(). //runtime error because the run time type is parent.
- Parent p = new Child ();
- Child c = (Child) p. //no error because runtime parent type is child


Parent
extends
Child
Parent p = new Child()
Upcasting
Upcasting
Child c = new Parent()
Compile time error
trying to downcast
Parent p = new Child();
Child c = (Child)p;
allowed cause p refers child
Downcasting

Instanceof

- Instanceof is a binary operator used to test if an object is of a given type. The result of the operation is either true or false. It's also known as type comparison operator because it compares the instance with type.
- If we apply the instanceof operator with any variable that has null value, it returns false.
- Child c = new Child();
  - c instanceof Child //true
  - c instanceof Parent //true
  - c instanceof Object. //true
- Child c1 = null;
  - c1 instanceof Child //false

Instanceof Operator and Downcasting

- Apart from testing object type, we can use it for object downcasting. However, we can perform downcasting using typecasting but it may raise ClassCastException at runtime.
- To avoid the exception, we use instanceof operator. Doing this helps to perform casting.
- When we do typecast, it is always a good idea to check if the typecasting is valid or not. instanceof helps us here. We can always first check for validity using instanceof, then do typecasting.
- Check the next slide

```java
class Parent{ }

public class Child extends Parent
{
    public void check()
    {
        System.out.println("Sucessfull Casting");
    }

    public static void show(Parent p)
    {
       if(p instanceof Child)
       {
           Child b1=(Child)p;
           b1.check();
       }
    }

    public static void main(String[] args)
    {

      Parent p=new Child();

      Child.show(p);

      }
}
```