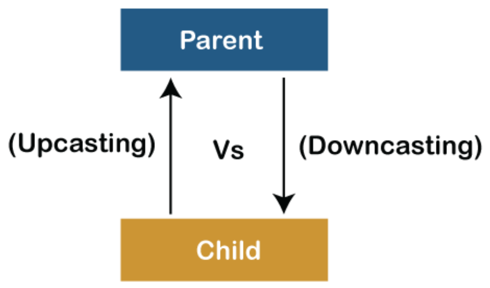
Day-17

Casting of Objects

Up casting and Down casting in Java



Up casting:

- Up casting refers to assigning a child class object to a parent class reference variable.
- The parent class reference can implicitly hold a child class object.

```java
Parent p = new Child(); // Upcasting
```

- From the parent class reference (**p**), we can only access members of the parent class.
- However, overridden methods from the child class will still be accessible because the object is created for the child class. This demonstrates runtime polymorphism.

Down casting:

- Down casting refers to assigning a parent class object to a child class reference variable.
- A child class reference **cannot** hold a parent class object directly, and attempting this leads to an error.

Invalid example:

```java
Child c = new Parent(); // Invalid: A child class reference holding a parent class obje
Child c = (Child) new Parent(); // Invalid: Results in java.lang.ClassCastException
```

- However, down casting is possible if the parent class reference is already holding a child class object. This requires explicit casting.

Valid example:

```java
Parent p = new Child(); // Upcasting
Child c = (Child) p;     // Downcasting
```

- After down casting, we can access both parent and child class members from the child class reference (**c**).
- Overridden methods in the child class can still be accessed from the reference, demonstrating runtime polymorphism.

Type Casting Syntax:

A b = (C) d;

Where:

- **A** is the type of the reference you are assigning to.
- **C** is the type you are casting to.
- **d** is the original reference.

Rules for Down casting:

Rule 1: Valid Conversion

The types of **d** and **C** must have a relationship, meaning they should either be in a parent-child relationship (either parent-to-child or child-to-parent), or they must be the same type.

Rule 2: Valid Assignment

The type **C** must either be the same type as **A** or a child of **A** in order for the assignment to be valid.

Rule 3: Underlying Object Type

The actual object type referred to by **d** must be either the same as **C** or a child of **C**.

Example:

```
class Parent {
}
class Child extends Parent {
}


Parent p = new Child(); // Upcasting
Child c = (Child) p;    // Downcasting
```

In this example:

- **Rule 1**: **p** is of type **Parent**, and **Child** has a relationship with **Parent** (child-to-parent relationship).
- **Rule 2**: The type **Child** is a child of **Parent**, so the assignment is valid.

- **Rule 3**: The underlying object type of **p** is **Child**, which matches the cast type, so no exception is thrown.

Example:

```java
class Fruit {}
class Apple extends Fruit {}
class Orange extends Fruit {}

public class TypeCastingObjectsRules {
    public static void main(String[] args) {

       // Example 1:

       Apple a=new Apple();
       Orange o=(Orange) a;  // Invalid as per Rule1


       // Example 2:

       Fruit f=new Apple();
       Orange o=(Apple) f; //Invalid as per Rule2


        // Example 3:
        Fruit f = new Apple();
        Orange o = (Orange) f; // Invalid as per Rule 3



        // Example 4:
        Fruit f = new Apple();
        Apple a = (Apple) f; //  valid

    }
}
```

Quiz: Casting Objects in Java (Upcasting & Downcasting)

Question 1:

What is **upcasting** in Java?

- A) Casting a child class object to a parent class type
- B) Casting a parent class object to a child class type
- C) Assigning primitive types to objects
- D) Type casting between unrelated classes

Answer: A) Casting a child class object to a parent class type

---

Question 2:

Which of the following is **true** about upcasting?

- A) It requires explicit casting
- B) It always results in a runtime exception
- C) It is implicit and safe
- D) It changes the actual object type

Answer: C) It is implicit and safe

Question 3:

What is **downcasting** in Java?

- A) Casting a child class object to a parent class type
- B) Casting a parent class object to a child class type
- C) Casting between different interface types
- D) Casting between unrelated classes

Answer: B) Casting a parent class object to a child class type

---

Question 4:

Why is **downcasting** considered unsafe?

- A) It can cause compilation errors
- B) It always results in data loss
- C) It requires explicit casting and can throw ClassCastException
- D) It prevents method overriding

Answer: C) It requires explicit casting and can throw ClassCastException

---

Question 5:

Which of the following is an example of **upcasting**?

```
class Animal {}
class Dog extends Animal {}
Animal animal = new Dog();
```

- A) Animal animal = new Dog();
- B) Dog dog = new Animal();
- C) Dog dog = (Dog) new Animal();
- D) None of the above

Answer: A) Animal animal = new Dog();

Question 6:

Which of the following requires **explicit casting**?

```
class Animal {}
class Dog extends Animal {}
```

- A) Animal animal = new Dog();
- B) Dog dog = (Dog) animal;
- C) Animal animal = (Animal) dog;
- D) Dog dog = new Dog();

Answer: B) Dog dog = (Dog) animal;

Question 7:

What happens if you attempt **downcasting** with an object that is not an instance of the target subclass?

- A) It compiles successfully but fails at runtime with ClassCastException
- B) It results in a compile-time error
- C) The object changes its type automatically
- D) Java automatically handles the conversion

Answer: A) It compiles successfully but fails at runtime with ClassCastException

---

Question 8:

Which of the following code snippets will **compile successfully** and execute without any exceptions?

```
class Animal {}
class Dog extends Animal {}
class Cat extends Animal {}

Animal animal = new Dog();
Dog dog = (Dog) animal;
```

- A) Dog dog = new Cat();
- B) Animal animal = new Dog();
- C) Dog dog = (Dog) new Cat();
- D) Cat cat = (Cat) animal;

Answer: B) Animal animal = new Dog();

Question 9:

Which of the following statements about **upcasting and downcasting** is correct?

- A) Upcasting allows access to subclass-specific methods
- B) Downcasting is implicit and safe
- C) Upcasting hides subclass-specific methods
- D) Downcasting is always safe at runtime

Answer: C) Upcasting hides subclass-specific methods