Shallow vs deep copy

explain shallow copy vs deep copy in java to answer interviewer and then give difference in table format

also explain

```
//Date.java

package com.sunbeam;

public class Date extends Object implements Cloneable {
    private int day;
    private int month;
    private int year;
    public Date() {
        // TODO Auto-generated constructor stub
    }
    public Date(int day, int month, int year) {
        this.day = day;
        this.month = month;
        this.year = year;
    }
    public int getDay() {
        return day;
    }
    public void setDay(int day) {
        this.day = day;
    }
    public int getMonth() {
        return month;
    }
    public void setMonth(int month) {
        this.month = month;
    }
    public int getYear() {
        return year;
    }
    public void setYear(int year) {
        this.year = year;
    }

    // this = dt1
    @Override
    protected Object clone() throws CloneNotSupportedException {
        Object temp = super.clone(); //shallow copy of instances
```

Printed using ChatGPT to PDF, powered by PDFCrowd HTML to PDF API.

```
            return temp;
        }
        @Override
        public String toString() {
            return "Date [day=" + day + ", month=" + month + ", year=" + year + "]";
        }

    }

    //Program.java

    package com.sunbeam;

    public class Program {

        public static void main(String[] args) throws CloneNotSupportedException {
            Date dt1 = new Date(1, 1, 2020);
            Date dt2 = (Date) dt1.clone( );

            System.out.println(dt1.toString());
            System.out.println(dt2.toString());

        }

    }
```

in this code , if we made changes in one object that changes will we reflected into other ?

Here's how you can frame your answer for an interview:

Shallow Copy vs Deep Copy in Java (Interview Answer)

In Java, **shallow copy** creates a new object, but it only copies the references of fields (not the actual objects they point to). That means both original and copied objects share the same reference for non-primitive fields.
On the other hand, **deep copy** creates a completely independent copy of an object along with its referenced objects. Changes in one object do not affect the other.

Difference Between Shallow Copy and Deep Copy

Aspect	Shallow Copy	Deep Copy
Definition	Copies the object but references point to the same inner objects.	Creates a completely independent object including copies of referenced objects.
Memory Sharing	Inner objects are shared between original and copy.	Each object has its own independent copy.

Aspect	Shallow Copy	Deep Copy
Implementation	Done using `Object.clone()` (default).	Requires overriding `clone()` and manually copying nested objects.
Performance	Faster (only references are copied).	Slower (creates full copies of inner objects).
Effect of Modification	Changes in inner objects reflect in both copies.	Changes in one object do not affect the other.

Explaining Your Code

Code:

```java
Object temp = super.clone(); // shallow copy of instances
```

- Here, `Date` only contains primitive fields (`int day`, `int month`, `int year`).
- **Primitives are copied by value** in both shallow and deep copy.
- So, even though you are doing a **shallow copy**, `dt1` and `dt2` are **independent objects**.

Behavior in Your Example:

```java
Date dt1 = new Date(1, 1, 2020);
Date dt2 = (Date) dt1.clone();
```

- If you modify `dt1.setDay(5);` → **it will not affect** `dt2`.
- That's because there are no object references inside `Date`, only primitives.

👉 **Answer for interviewer**:
In this specific code, changing one object will **not reflect in the other** because all fields are primitives. If the class had a reference type (e.g., another `Person` object inside `Date`), then shallow copy would share the reference, and changes in one would reflect in the other.

Do you want me to also extend this example by **adding a reference field (like Person owner inside Date)** to clearly show where shallow copy causes shared references and deep copy solves it?

> but you say in shallow copy both object points to same reference so changes in one should change another ?

Good doubt 👍 Let me clear the confusion.

👉 The rule is:

- **Shallow copy**:

    - The **top-level object** is duplicated.
    - But if it has **reference-type fields**, only the *references* are copied, not the actual objects.
    - So both objects share the same referenced instance.
    - Changes in that shared object will reflect in both.
- **Deep copy**:
    - Both top-level and nested objects are cloned.
    - No sharing of references.
    - Changes won’t reflect.

Why in your `Date` class shallow copy looks like deep copy?

Because your class has only **primitive fields** (`int day, month, year`).

- Primitives (`int`, `double`, `char`, etc.) are **copied by value**, not by reference.
- So even with a shallow copy, each object (`dt1`, `dt2`) gets its own independent `day`, `month`, and `year`.

That’s why:

```java
dt1.setDay(5);
System.out.println(dt2.getDay()); // Still 1, not 5
```

Example with reference field (where shallow copy problem appears):

```java
class Person {
    String name;
    public Person(String name) { this.name = name; }
}

class Employee implements Cloneable {
    int id;
    Person person;  // reference field

    public Employee(int id, Person person) {
        this.id = id;
        this.person = person;
    }

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return super.clone(); // shallow copy
    }
}

public class Test {
    public static void main(String[] args) throws Exception {
        Employee e1 = new Employee(101, new Person("Uday"));
        Employee e2 = (Employee) e1.clone();

        e1.person.name = "Changed"; // modifying nested object

        System.out.println(e1.person.name); // Changed
        System.out.println(e2.person.name); // Changed (shared reference!)
    }
}
```

👉 Here, shallow copy makes both `e1` and `e2` share the same `Person`.
👉 If you did deep copy (manually cloning `Person` too), then `e2` would remain unaffected.

✅ So, to your question:

- **Yes**, in shallow copy *references* are shared.
- In your `Date` example, there are **no reference fields**, only primitives, so both objects behave independently.

Do you want me to **modify your `Date` class to include a reference field (like `Person createdBy`)** so you can directly see shallow vs deep copy in action?

so give me proper answer now to tell interviewer

Got it 👍 Here's how you can frame a **clear, interview-ready answer**:

Answer: Shallow Copy vs Deep Copy in Java

In Java,

- **Shallow Copy**:
  It creates a new object, but copies only the **primitive fields** by value and **reference fields** by reference. So, if the object has references to other objects, both copies will point to the same inner objects. Changes in the referenced object will reflect in both.
- **Deep Copy**:
  It creates a completely independent object along with copies of all nested objects. This means any modification in one object does not affect the other.

Difference Table

Aspect	Shallow Copy	Deep Copy
Definition	Copies primitive fields and references (same inner object).	Copies both primitive fields and creates independent copies of referenced objects.
Memory Sharing	Inner objects are shared.	Inner objects are not shared.
Effect of Changes	Changes in referenced objects affect both.	Changes do not affect the other object.
Implementation	Default `clone()` (super.clone()).	Override `clone()` and manually clone referenced fields.
Performance	Faster (less memory usage).	Slower (uses more memory).

Explaining the Given Code

```java
Date dt1 = new Date(1, 1, 2020);
Date dt2 = (Date) dt1.clone();
```

- Your `Date` class only has **primitive fields** (`int day, month, year`).
- In Java, primitives are copied **by value**.
- So even though you used `super.clone()` (shallow copy), both `dt1` and `dt2` have **independent copies** of day, month, and year.
- ✅ Therefore, if you change `dt1.setDay(5)`, it will **not affect** `dt2`.

👉 But if the class had a reference field (e.g., a `Person` object inside `Date`), then shallow copy would cause both objects to share the same reference, and changes in one would reflect in the other. In that case, you'd need a deep copy to make them independent.

✨ **Short Interview-style Answer**:
"In Java, shallow copy copies primitive fields by value but reference fields by reference, so nested objects are shared. Deep copy creates a full independent copy, including referenced objects. In my `Date` example, since it has only primitives, changes in one object won't affect the other. But if it had reference fields, shallow copy would share them, while deep copy would not."

Do you want me to also give you a **1-liner difference** (super short) in case interviewer asks for just a quick definition?