Bulleted Notes on Copying Objects in Java with Examples

1. Introduction

- Copying objects is a core concept in Java.
- Two primary approaches:
  - **Shallow Copying**
  - **Deep Copying**
- Cloning is another method using the `clone()` method.

2. Shallow Copy

- Creates a new object that shares references with the original object.
- Changes in one object affect the other.
- Default behavior when assigning one object to another or using `clone()` without modifications.

Example:

```
class ShallowCopy {
   private int[] data;

   public ShallowCopy(int[] values) {
      data = values; // Reference assignment
   }

   public void printData() {
      for (int value: data) {
         System.out.print(value + " ");
      }
      System.out.println();
   }
}

public class Main {
   public static void main(String[] args) {
      int[] values = {1, 2, 3};
      ShallowCopy obj1 = new ShallowCopy(values);
      ShallowCopy obj2 = obj1; // Shallow Copy

      obj1.printData();
      obj2.printData();

      values[0] = 10; // Modify original array

      obj1.printData();
      obj2.printData();
   }
}
```

Output:

```
1 2 3
1 2 3
10 2 3
10 2 3
```

3. Deep Copy

- Creates a completely independent object with its own memory.
- Changes in one object do not affect the other.
- Requires manually copying primitive and reference-type fields.

Example:

```
class DeepCopy {
    private int[] data;

    public DeepCopy(int[] values) {
        data = new int[values.length];
        for (int i = 0; i < values.length; i++) {
            data[i] = values[i];
        }
    }

    public void printData() {
        for (int value : data) {
            System.out.print(value + " ");
        }
        System.out.println();
    }
}

public class Main {
    public static void main(String[] args) {
        int[] values = {1, 2, 3};
        DeepCopy obj1 = new DeepCopy(values);
        DeepCopy obj2 = new DeepCopy(values); // Deep Copy

        obj1.printData();
        obj2.printData();

        values[0] = 10; // Modify original array

        obj1.printData();
        obj2.printData();
    }
}
```

Output:

```
1 2 3
1 2 3
1 2 3
1 2 3
```

4. Cloning

- Built-in `clone()` method creates a copy.
- The class must implement the `Cloneable` interface.
- By default, shallow copy is performed.
- Deep cloning can be implemented manually inside the `clone()` method.

Example:

```
class Cloning implements Cloneable {
    private int[] data;

    public Cloning(int[] values) {
        data = values;
    }

    public void printData() {
        for (int value : data) {
            System.out.print(value + " ");
        }
        System.out.println();
    }

    @Override
    public Cloning clone() throws CloneNotSupportedException {
        Cloning cloned = (Cloning) super.clone();
        cloned.data = data.clone(); // Deep Copy for array
        return cloned;
    }
}

public class Main {
    public static void main(String[] args) {
        int[] values = {1, 2, 3};
        Cloning obj1 = new Cloning(values);

        try {
            Cloning obj2 = obj1.clone();

            obj1.printData();
            obj2.printData();

            values[0] = 10; // Modify original array

            obj1.printData();
            obj2.printData(); // No change due to deep copy
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
1 2 3
1 2 3
10 2 3
1 2 3
```

6. When to Use It

- **Shallow Copy:**
  - When memory optimization is needed.
  - When shared data references are acceptable.
- **Deep Copy:**
  - When data independence is required.
  - For immutable objects.
- **Cloning:**
  - When creating object copies frequently.
  - If custom copy behavior is needed.