Cloneable Interface in Java

Introduction:

A cloneable interface is a marker interface i.e. it is empty and without any field. The object class has a clone method which overridden and monitored by this interface. This clone method is user-defined and the variables present in it should be static.

A cloneable interface creates a separate copy of the invoking object without using a new operator and both will have different references. The new memory space will be occupied by the object created by the clone method. This interface was introduced in JDK 1.0 in Java.lang package. it reduces the time and effort of the developers to copy the object.

Declaration:-

```
Public interface Cloneable
```

Example:

```
public class Exam implements Cloneable {
    private int i = 2;
 public object clone() {
        try {  super.clone(); }
        catch(Exception e){ return null; }
        }
    }
    public static void main(String args[]) {
   Exam e1 = new Exam();
    e1.i = 5;
    Exam e2 = (Exam)e1.clone();

      System.out.println("e2.i");
    }
}
```

Output:

```
5
```

If we do not use a cloneable interface then the program will throw a CloneNotSupported exception. The overridden clone method can also throw this exception if the object is not possible to be cloned.

Example

```
public class Employee {
      String emp_name = null;
      int emp_ID = 0;

      Employee(){}

      Employee(String emp_name, int emp_ID){
          this.Emp_name = name
          this.Emp_ID = id

      }
public static void main(String[] args)
    {

        Employee e1 = new Employee("Apoorva", 711);


        Employee e2 = e1.clone();
    }
}
```

Output:

```
Incompatible types: Object cannot be converted to Employee
         Employee E2 = E1.clone();
                                ^
```

To resolve this issue we use the interface cloneable and a code to handle the exception.

```
Class O implements Cloneable {
   int i;
   String s;

   public O(int i, String s)
   {
      this.i = i;
      this.s = s;
   }
   protected Object clone()
      throws CloneNotSupportedException
   {
      return super.clone();
     }
}

public class Test {
   public static void main(String[] args)
      throws CloneNotSupportedException
     {
O a = new O(45, "Freelancer");

      O b = (O)a.clone();

          System.out.println(b.i);
          System.out.println(b.s);
     }
}
```

Output:

```
45
Freelancers
```

Cloneable Interface and Deep Copying:

Java class does not support cloning and the clone() method's default implementation raises the CloneNotSupportedException exception. The clone() method should be overridden. Keep in mind that you must make it public and that super.clone() must be the method's first call. If classes need cloning then implementation of the Cloneable marker interface is necessary. Since the default implementation of Object. clone only performs a shallow copy, classes must also override clone to provide a custom implementation when a deep copy is desired.

For Deep Cloning:

It can be implemented by java serialization. This will give the desired cloning.

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
ObjectOutputStream oos = new ObjectOutputStream(baos);
oos.writeObject(this);
ByteArrayInputStream bais = new ByteArrayInputStream(baos.toByteArray());
ObjectInputStream ois = new ObjectInputStream(bais);
Object deepCopy = ois.readObject();
```

Conclusion:

As we previously discussed, there are numerous ways to duplicate an object. One of the most effective of these is the cloneable interface in Java. Cloning appears to be a fairly basic capability, but it seeks to save a lot of processing and time in larger projects. The only issue is that it needs to be used correctly and only when necessary to effectively benefit from it. Before using the Cloneable interface in Java, developers must first comprehend how it works properly, its ramifications, and where issues could occur.