**java.lang package – Part-13**

* **Clone**

The process of creating exactly duplicate object is called Cloning.

The main purpose of cloning is:

To maintain a backup copy and to preserve the state of an object.

We can perform cloning by using clone method of Object class.

protected native Object clone() throws CloneNotSupportedException

* **Creating a cloned object:**

class CloneTest implements Cloneable{

int i = 10;

int j = 20;

public static void main(String[] args) throws CloneNotSupporedException{

CloneTest t1 = new CloneTest();

CloneTest t2 = (CloneTest)t1.clone();

t2.i = 888;

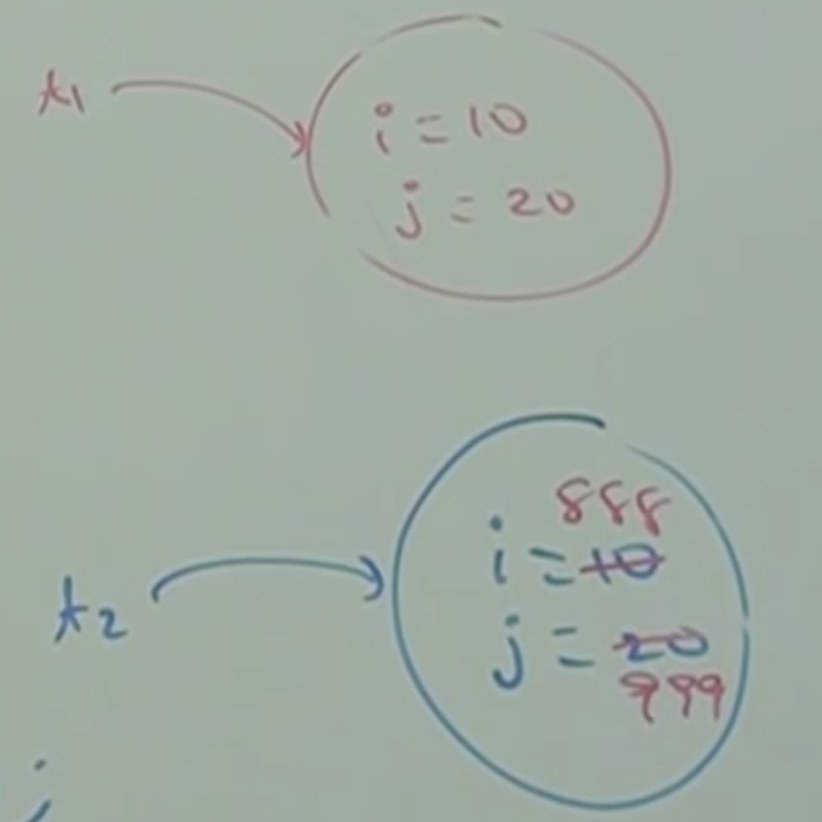
t2.j = 999;

System.out.println(t1.i+”….”+t1.j);

}

}

**Output: 10…20**

****

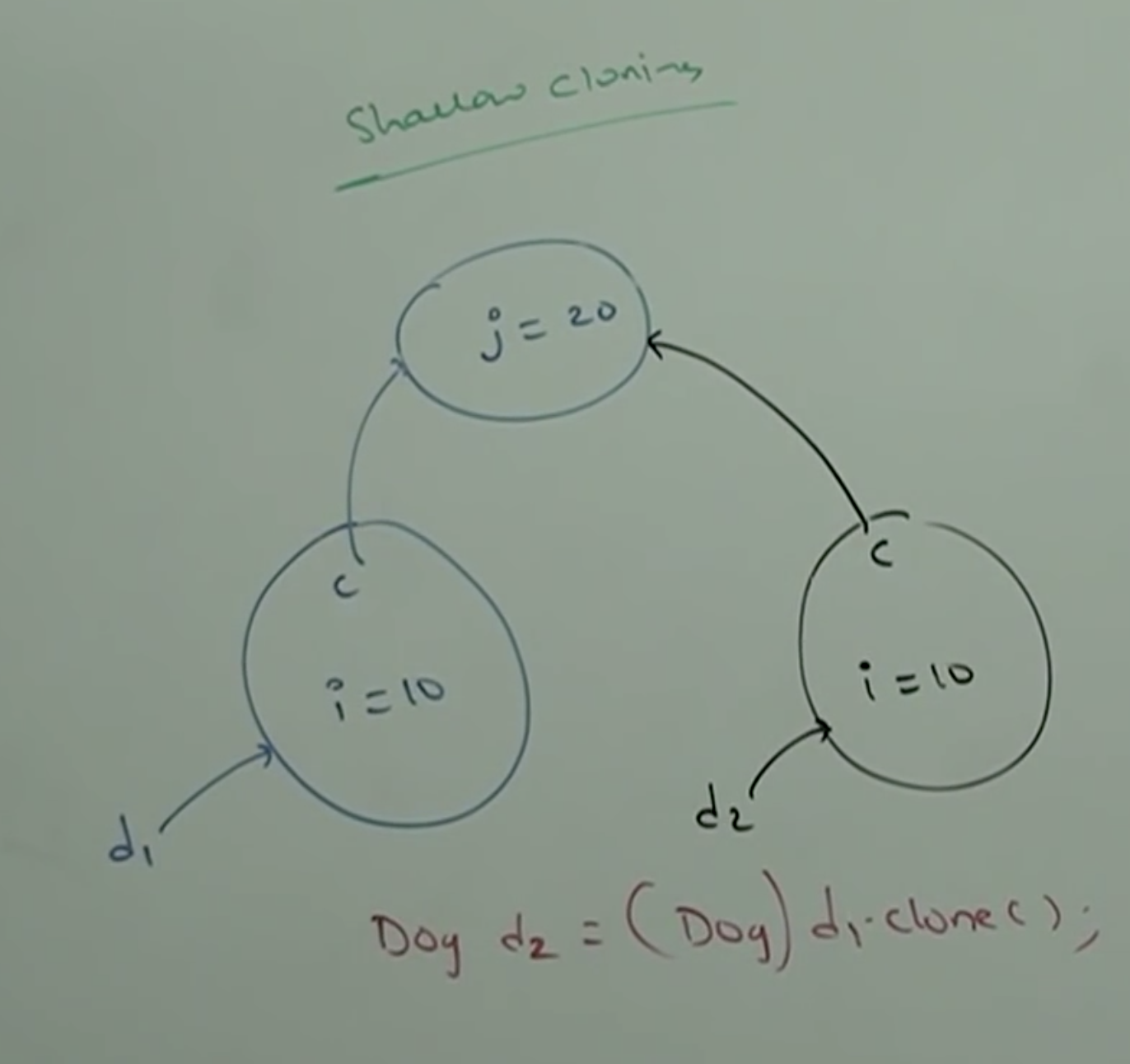
We can perform cloning only for Cloneable objects. An object is said to be cloneable, if and only if the corresponding class implements Cloneable interface.

Cloneable interface present in java.lang package and it doesn’t contain any methods. It is a marker interface.

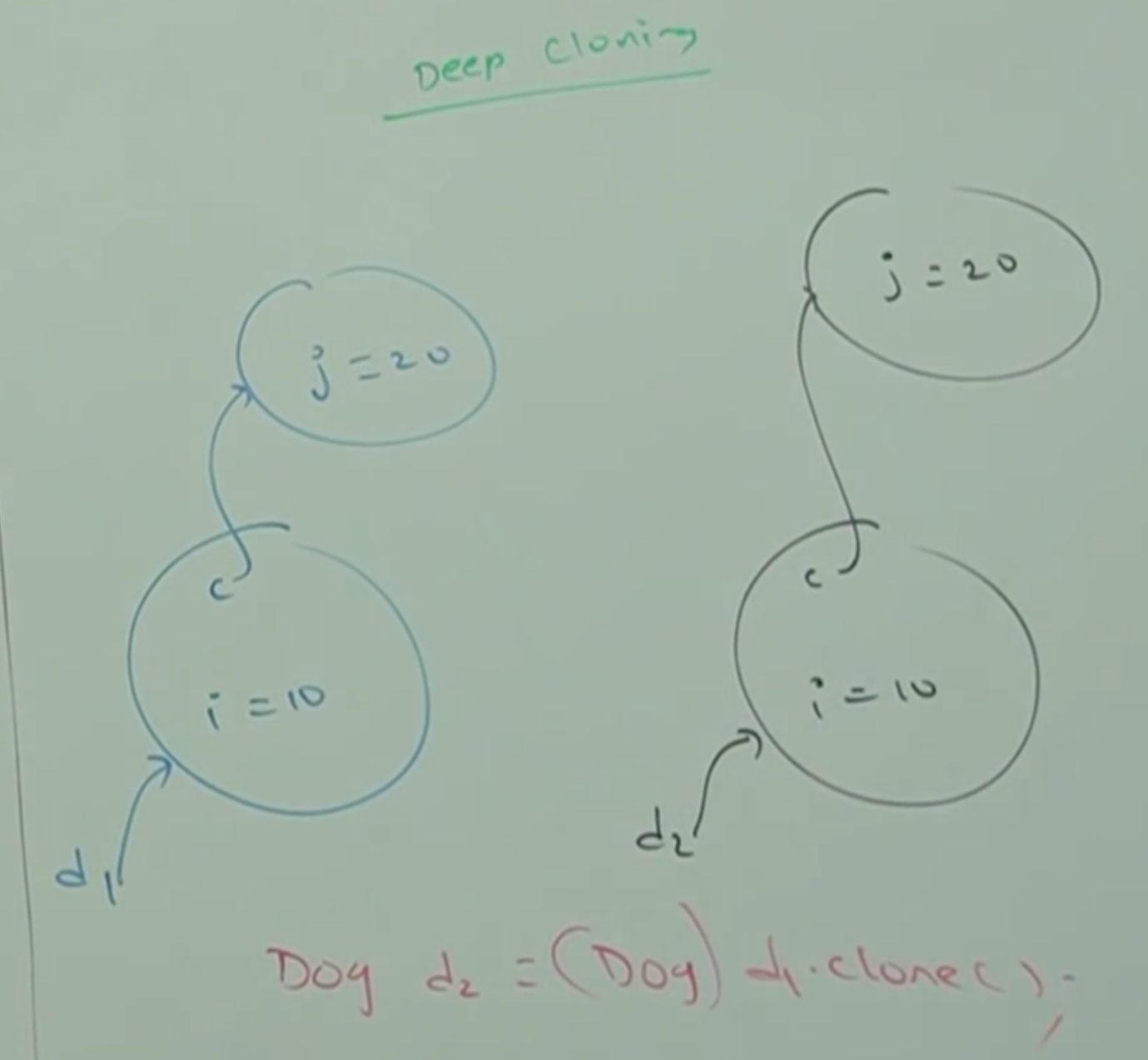
If we are trying to perform cloning for non-cloneable objects then we will get runtime exception saying: CloneNotSupportedException

* **Shallow Cloning VS Deep Cloning:**

Shallow Cloning:



Deep cloning:



* **Shallow Cloning:**

The process of creating bit-wise copy of an object is called shallow cloning.

If the main object contains primitive variables then exactly duplicate copies will be created in the cloned object.

If the main object contains any reference variable then corresponding object won’t be created just duplicate reference variable will be created pointing to old contained object.

Object class clone method meant for shallow cloning.

Example:

class Cat{

int j;

Cat(int j){

this.j = j;

}

}

class Dog implements Cloneable{

Cat c;

int i;

Dog(Cat c, int i){

this.c = c;

this.i = i;

}

public Object clone() throws CloneNotSupportedException {

return super.clone();

}

}

class ShallowCloning{

public static void main(String[] args) throws CNSE{

Cat c = new Cat(20);

Dog d1 = new Dog(c, 10);

System.out.println(d1.i+”…”+d1.c.j);

Dog d2 = (Dog)d1.clone();

d2.i = 888;

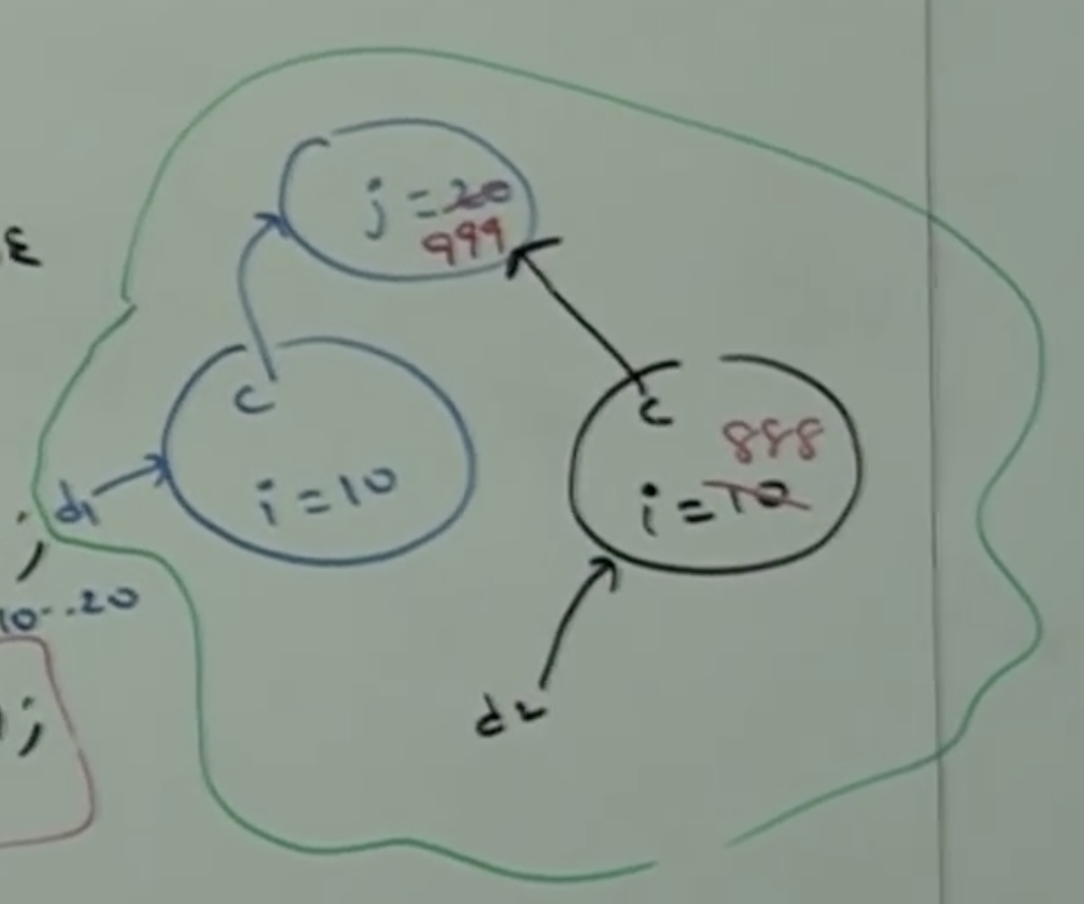
d2.c.j = 999;

System.out.println(d1.i+”…”+d1.c.j);

}

}

Output: 10…999



In shallow cloning by using cloned object reference if we perform any change to the contained object, then those changes will be reflected to the main object.

To overcome this problem, we should go for deep cloning.

* **Deep Cloning:**

The process of creating exactly duplicate independent copy including contained object is called deep cloning.

In deep cloning if the main object contains any primitive variables then in the cloned object duplicate copies will be created.

If the main object contains any reference variable, then the corresponding contained also will be created in the cloned copy.

By default Object class clone() method is meant for shallow cloning. But we can implement deep cloning explicitly by overriding clone() method in our class.

Example:

class Cat{

int j;

Cat(int j){

this.j = j;

}

}

class Dog implements Cloneable{

Cat c;

int i;

Dog(Cat c, int i){

this.c = c;

this.i = I;

}

public Object clone()throws CloneNotSupportedException{

Cat c1 = new Cat(c.j);

Dog d1 = new Dog(c1, i);

return d1;

}

}

class DeepCloningDemo{

public static void main(String[] args) throws CNSE{

Cat c = new Cat(20);

Dog d = new Dog(c, 10);

System.out.println(d.i+”…”+d.c.j); // 10…20

Dog d2 = (Dog)d.clone();

d2.i= 888;

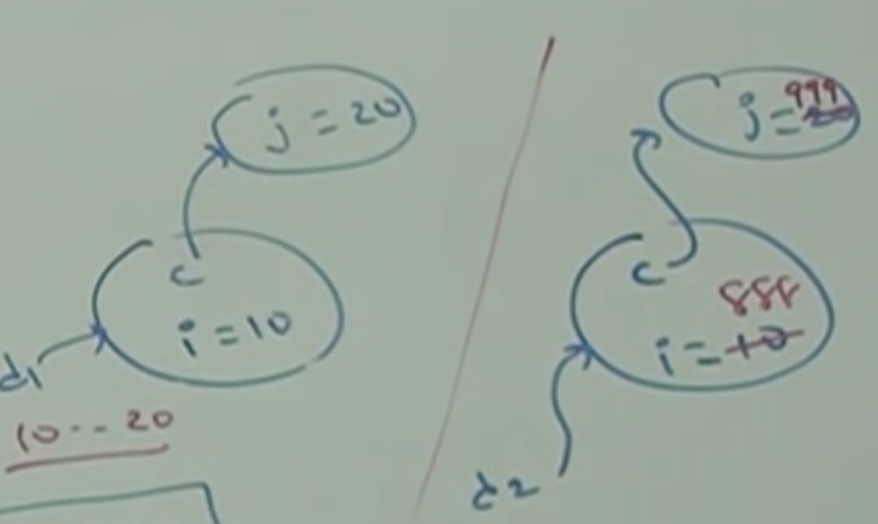
d2.c.j = 999;

System.out.println(d.i+”…”+d.c.j); // 10…20

}

}

Output: 10…20



By using cloned object reference if we perform any change to the contained object then those changes won’t be reflected to the main object.

* **Which cloning is best?**

If object contains only primitive variables then shallow is the best choice.

If object contains reference variables then deep cloning is the best choice.