SCHULICH School of Engineering

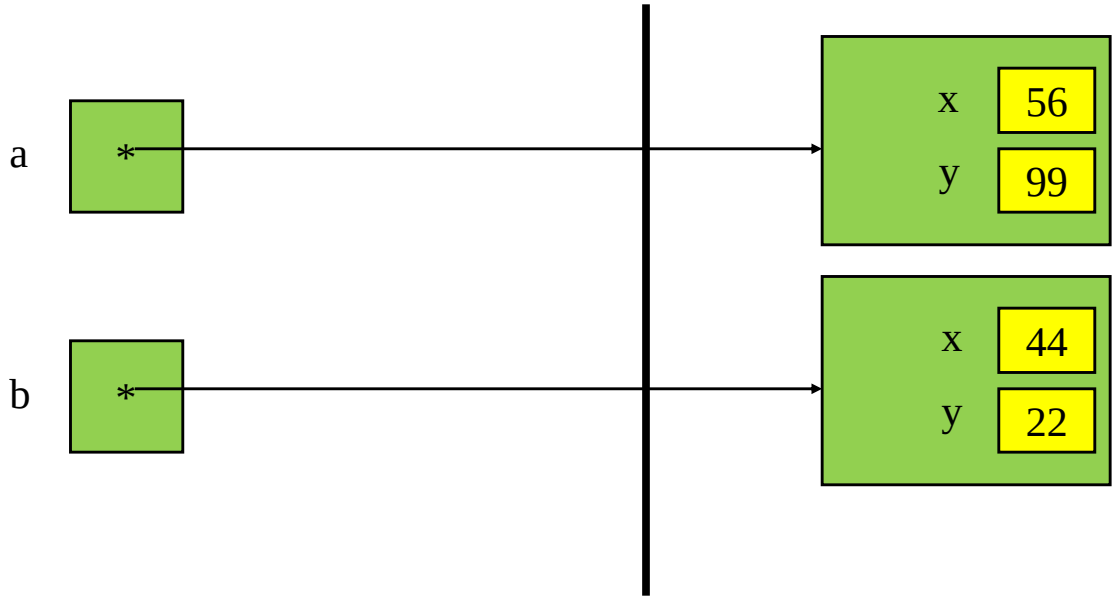
ENSF 593/594

10 – Copying Objects

Copying Objects

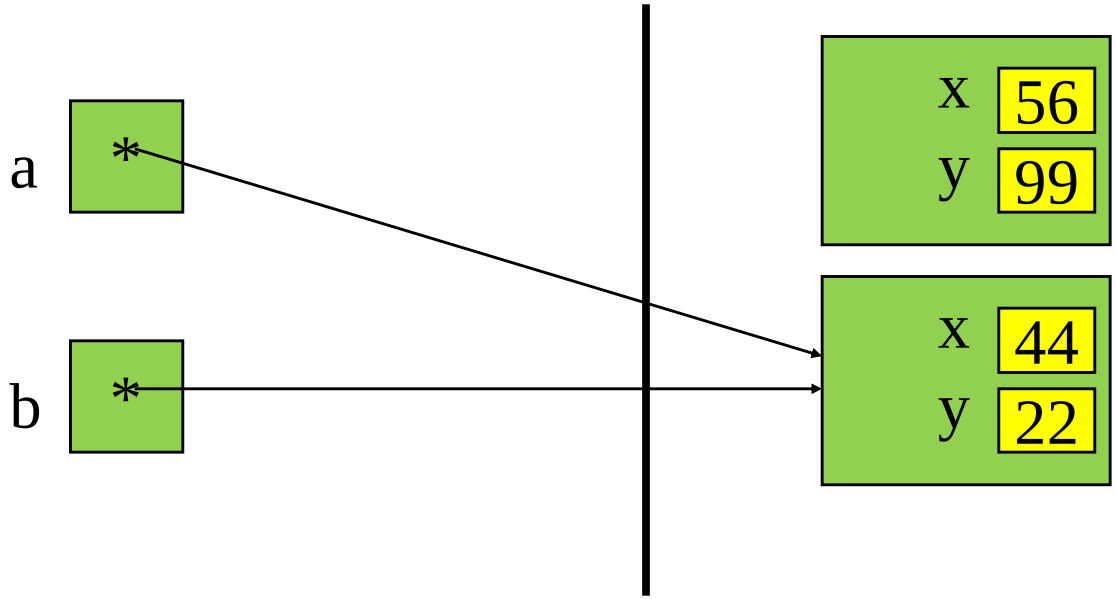
Point a = new Point(56, 99);

Point b = new Point (44, 22);



Copying Objects

a = b;



Cloning Objects (continued)

- If a class contains a reference to another object (i.e. new operator is used in a constructor), the "part" objects **must be** cloned in the clone() method of the container class.

Clone Method

- In Java you **MUST** use clone method to make an actual copy of an object. To create a clone method for a class, you need to:
  1. implement the **Cloneable interface**. This is a marker interface (an empty interface).
  2. redefine the **clone()** method
  3. deal with the **CloneNotSupportedException**

- Note: **Marker interfaces** such as **Cloneable**, **Serilizeable**,and **Remote**, do not have any method. They are used as **markers** to give a signal to the compiler or JVM

Class Exercise

Example

- In an application we have three classes as follows:
  - Point
  - Shape
  - Rectangle
- Each shape is supposed to have a point, and we need to make copies of shapes shape or rectangle.

Clone Method

```
public class Point implements Cloneable {
    …

    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }

    …
}
```

Class Point implements the Cloneable interface

throws this exception if any errors occur in this method

Using Clone Method

```
public static void main(String args[])
{
    Point a, b;
    a = new Point(3.0, -5.0);

    try {
        b = (Point)a.clone();
        System.out.println("b: x = " + b.x() +
                                   "y = " + b.y());
    }
    catch(CloneNotSupportedException e) {
        System.out.println("Can't clone Point a");
    }
}
```

Call to the clone method must happen within a try block, followed by one or more catch blocks.

Cloning and Aggregation

```
public class Shape implements Cloneable {
    Point origin;

    public Shape() {
        origin = new Point();
    }

    public Object clone() throws CloneNotSupportedException {
        Shape obj = (Shape)super.clone();

        obj.origin = (Point)origin.clone();
        return obj;
    }
}
```

the origin point is created using new operator

Needs also the cast operator

Call to the clone method in the Point class

Cloning and Inheritance

- Lets assume class Rectangle extends from Shape and contains an object of class Color:

```
public class Rectangle extends Shape implements Cloneable {
    protected double with, length;
    Color c;
    public Rectangle(double x, double y, double w, double l )
    {
        super (x, y);
    }

    public Object clone() throws CloneNotSupportedException {
        Rectangl obj = (Rectangle)super.clone();
        obj.c = (Color) c.clone();
        return obj;
    }
}
```

Some Review Questions

- Explicit vs. implicit sub classing.
- Access control in java?
- What is an abstract method
- What are the impacts of an abstract method on your class.
- What is cloning? Why, and when it is needed.

Review Questions

- Consider the following example of Java classes. How do you define a clone method?

```
class A{
private int x;
private B myb
…
}
```

```
class B{
private int y;
private C myc;
…
}
```

```
class C{
private int  z;

…
}
```

Example

```
public class D {
	static public void main(String[] args) {
		A a1, a2 = null;
		a1 = new A();
		a1.setx(5);
		try {
			a2 = (A) a1.clone();
		} catch (CloneNotSupportedException e) {
			System.out.println("Can't clone Point a");
		}

		a1.setx(10);

		System.out.println(a1.getx());  //Outputs 10
		System.out.println(a2.getx());  //Outputs 5
	}

}
```

Example Cont'd

```
public class C implements Cloneable {
    private int z;

    public Object clone() throws CloneNotSupportedException {
       return super.clone();
    }

    public void setz(int a) {a = z;}

    public int getz() { return z; }

}
```

```
public class B implements Cloneable {
        private int y;
        private C myc;

        public B ()
        {
                myc = new C ();

        }
        public Object clone() throws CloneNotSupportedException {
                B temp = (B)super.clone();
                temp.myc = (C)myc.clone();
                return temp;
                // point one
        }

        public void sety (int a) { y = a;}

        public int gety () {return y;  }
}
```

Example Cont'd

```
class A implements cloneable {
private int x;
private B myb

public Object clone() throws CloneNotSupportedException {
    ??? temp = (???)super.clone();
    temp.??? = (???)???.clone();
    return temp;
}


}
```

Example Cont'd – Class A Solution

```
public class A implements Cloneable {

        private int x;
        private B myb;

        public A ()
        {
                x = 0;
                myb = new B();
        }

        public Object clone () throws CloneNotSupportedException
        {
                A temp = (A)super.clone();
                temp.myb = (B) myb.clone();
                return temp;
        }

        public int getx() { return x;}
        public void setx(int a) { x = a;}
}
```

Question

- What if C is extended from B and B is extended from A?

```
class C {
    protected int x;

…
}
```

```
class B extends C{
    protected int y;
…
}
```

```
class A extends B{
    protected  int  z;
…
}
```

Solution

```
public class D {
      static public void main(String[] args) {
            A a1, a2 = null;
            a1 = new A();
            a1.setz(5);
            try {
                  a2 = (A) a1.clone();
            } catch (CloneNotSupportedException e) {
                  System.out.println("Can't clone Point a");
            }

            a1.setz(10);

            System.out.println(a1.getz());  //Outputs 10
            System.out.println(a2.getz());  //Outputs 5
      }

}
```

Solution Cont'd

```
public class C implements Cloneable{

   protected int x;

   public Object clone () throws CloneNotSupportedException
   {
      return super.clone();
   }

}
```

Solution Cont'd

```
public class B extends C{

    protected int y;

    public Object clone () throws CloneNotSupportedException
    {
        return super.clone();
    }
}
```

Solution Cont'd

```
public class A extends B {

    protected  int  z;

    public Object clone () throws CloneNotSupportedException
    {
        return super.clone();
    }

    public void setz(int i) {z = i;}

    public int getz() {return z; }


}
```