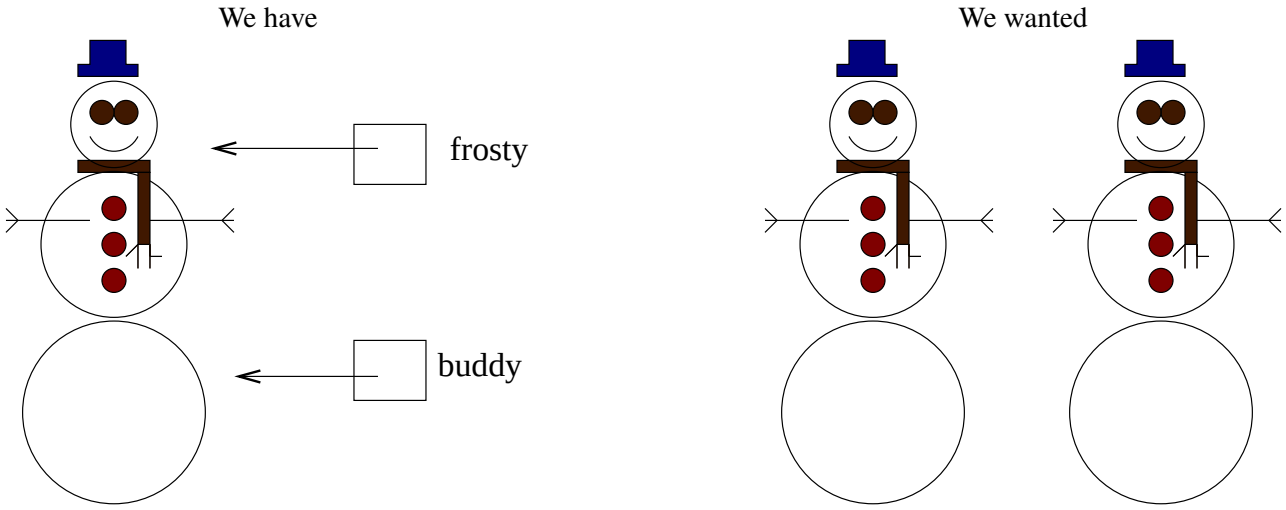
1 Why do we need copy constructors?

Suppose we have an instance of a Snowman and we want to duplicate it. The Snowman's member data are references to its hat, eyes, buttons, and scarf. We might just copy as we would for integers.

```
Snowman frosty = new Snowman();
Snowman buddy = frosty;
```



We ended up with one snowman with multiple personality disorder. We should have used a special constructor called a *copy constructor*.

2 What is a copy constructor ?

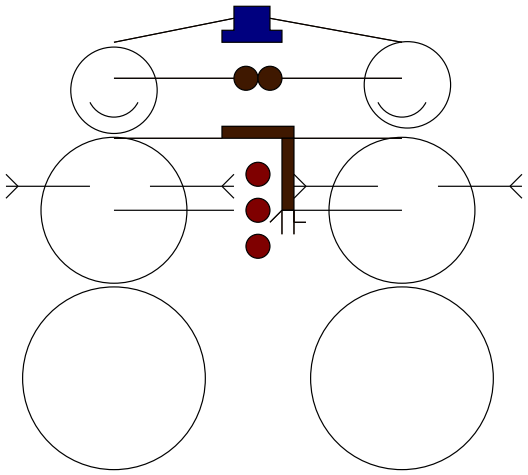
```
public class Snowman {
    int numSnowballs;
    Scarf scarf;
    Hat hat;
    Button[] buttons;
    Eye[] eyes;
    int[] snowballWidth;

    public Snowman() { ... }                  // default constructor
    public Snowman(String name) { ... }       // constructor
    public Snowman(Snowman old) { ... }       // copy constructor
}
```

A copy constructor for a Snowman takes the original Snowman as a paramter and copies all of its member data.

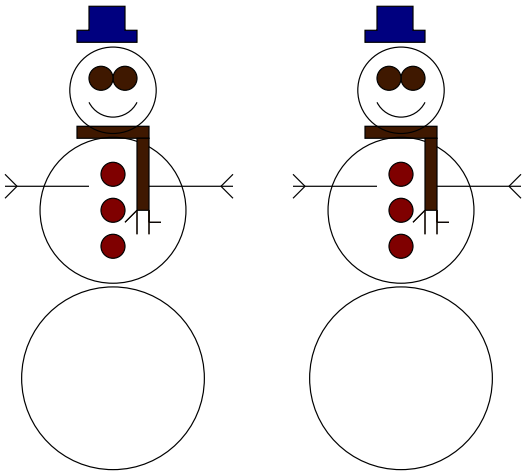
```
public Snowman(Snowman old) {
    numSnowballs = old.numSnowballs;
    scarf = old.scarf;
    hat = old.hat;
    buttons = old.buttons;
    eyes = old.eyes;
}
```

Now we get the following picture.

We have



We wanted



We ended up with a pair of conjoined twins who share their hat, scarf, eyes, and buttons. The problem is that we made a *shallow copy* of each member data. We needed to *deep copy* all of the references and arrays.

3 How to deep copy the member data

Separate the member data into three catagories:

- primitive data
- reference data (objects)
- arrays

3.1 Deep copying primitive data

Since primitive data stores its value in the box of memory with it's name, every copy is a deep copy. We might need to use `get` methods on the old Snowman if its member data is private.

```
public class Snowman {
    int numSnowballs;

    public Snowman(Snowman old) {
        numSnowballs = old.numSnowballs;
    }
}
```

3.2 Deep copying reference data

Since reference data contains a reference to the actual object, we need to invoke the copy constructor on each one.

```
public class Snowman {
    Hat hat;
    Scarf scarf;

    public Snowman(Snowman old) {
        hat = new Hat(old.hat);
        scarf = new Scarf(old.scarf);
    }
}
```

If we didn't deep copy these using their respective copy constructors, we'd get a shared hat and scarf.

3.3 Deep copying arrays

You need to first duplicate the array and then deep copy the contents. Our snowman has an array to hold the Buttons down his front and an array of eyes. It also has an array of integers which state the width of each snowball.

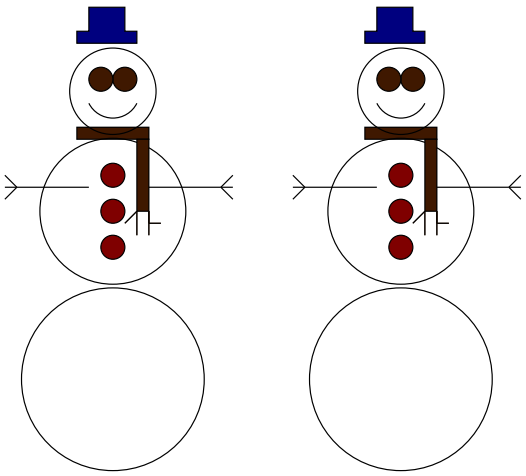
```
public class Snowman {
    Button[] buttons;
    Eye[] eyes;
    int[] snowballWidth;

    public Snowman(Snowman old) {
        // deep copy both arrays
        buttons = new Button[old.buttons.length];
        eyes = new Eye[old.eyes.length];
        snowballWidth = new int[old.snowballWidth.length];

        // copy the contents of the primitive array
        for (int i=0; i<snowballWidth.length; i++)
            snowballWidth[i] = old.snowballWidth[i];

        // deep copy the contents of the reference arrays
        for (int i=0; i<buttons.length; i++)
            buttons[i] = new Button(old.buttons[i]);

        for (int i=0; i<eyes.length; i++)
            eyes[i] = new Eye(old.eyes[i]);
    }
}
```

If we didn't deep copy these, we'd get a shared set of buttons, eyes, and widths.

4 How do we use a copy constructor?

```
Snowman frosty = new Snowman();
Snowman buddy = new Snowman(frosty);
```

Success!



5 The clone method

The clone method of a Java Object makes a shallow copy of the Object and returns it. In order to use it, it must be specifically implemented and you must be prepared to catch the exceptions it throws. If this is what you want, use it. Otherwise be prepared to write a copy constructor.

```
public class Snowman implements Cloneable {
    public Object clone() {
        try {
            Snowman cloned = (Snowman) super.clone();
            cloned.hat = (Hat) hat.clone();
            cloned.scarf = (Scarf) scarf.clone();
            // handle the arrays much as seen above
        }
        catch (CloneNotSupportedExceptions e) {
            return null;
        }
    }
}
```