Assigning objects and copying constructors
Class User {
    public String name;
    public int age;
    public User(String str, int a) {
        Name = str; age = a;
    }
}

Class Test {
    public static void main(String[] args) {
        User u1 = new User("Ralph", 112);
        System.out.println(u1.name); // Ralph
        User u2 = u1;
        u2.name = "Bond";
        System.out.println(u1.name); // Bond
    }
}
Class User {
    public String name;
    public int age;
    public User(String str, int a) {
        Name = str; age = a;
    }
}

Class Test {
    public static void main(String[] args) {
        User u1 = new User("Ralph", 112);
        System.out.println(u1.name); // Ralph
        User u2 = u1;
        u2.name = "Bond";
        System.out.println(u1.name); // Bond
    }
}
Class User {
    public String name;
    public int age;
    public User(String str, int a) {
        Name = str; age = a;
    }
}

Class Test {
    public static void main(String[] args) {
        User u1 = new User("Ralph", 112);
        System.out.println(u1.name); // Ralph
        User u2 = u1;
        u2.name = "Bond";
        System.out.println(u1.name); // prints Bond
    }
}

But what if we want u2 to reference a copy of what u1 references?
Sometimes a copy or *clone* of an object is desired

- Even though Java only copies references, it is sometimes desirable that an assignment put a reference to a new object in the left hand side (LHS) variable.

- *Cloning* is the Java mechanism for accomplishing this.

- The class for objects to be cloned *must* implement the interface *Clonable*.

- Invoking the *clone* method on an object produces a clone of the object.
• Even though X is a very simple class, cannot invoke clone because it does not implement cloneable.

• We can duplicate X and its state, as shown below:

```java
class X {
    int n;
    X() {n = 3;}
    X(in n) {
        this.n = n;
    }
    getN( ) {
        return n;
    }
}
X xobj = new X(4);
Y yobj = new X(xobj.getN( ));
```

• But . . .

  • What if X was a hashmap or something else complicated?
  • What about private fields?
  • Encapsulation implies objects should copy themselves.
class X implements Cloneable {
    int n;
    X() {n = 3;}
    X(in n) {
        this.n = n;
    }
    int getN() {
        return n;
    }
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}

X xobj = new X(4);
X xobjClone = (X) xobj.clone();
The `Object clone` function 
(super.clone( )) makes a byte-by-byte 
copy of the object referenced by `xobj` 
and returns a reference to it.

```java
class X implements Cloneable {
    int n;
    X() {n = 3;}
    X(in n) {
        this.n = n;
    }
    int getN() {
        return n;
    }
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}

X xobj = new X(4);
X xobjClone = (X) xobj.clone();
```
class X implements Cloneable {
    int n;
    X() {n = 3;}
    X(in n) {
        this.n = n;
    }
    int getN() {
        return n;
    }
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}

X xobj = new X(4);
X xobjClone = (X) xobj.clone();
The interface cloneable is not a normal interface

- *Cloneable* is empty -- a class implementing it doesn't have to actually implement anything (but can if it wants)
- but if a `public Object clone()` method is not implemented an error will result if an attempt is made to clone the object.
- *Implements Cloneable* is a signal to the *Object* class that it is ok for *Object’s clone* to clone this object w/a byte for byte copy.
- This leverages the fact that *Object* is not a normal class. It, and some other systems classes, perform functionality not expressible in Java.
Calling clone

import java.util.*;
class X implements Cloneable {
    public int n;
    public X() {n=3;}
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}

try {
    xobj_clone = (X) xobj.clone();
} catch (CloneNotSupportedException e) { ... }

...
class X implements Cloneable {
    public int[] arr = new int[5]
    public X() {
        Random ran = new Random( );
        int i = 0;
        while (i < 5) {
            arr[i++] = ran.nextInt(10);
        }
    }
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}

X xobj = new X();
Consider this class X

X xobj = new X();

X xobjClone = xobj.clone();

The standard clone will do this

What if we also wanted arr cloned?
What if we want this?

```java
X xobj = new X();
X xobjClone = xobj.clone();
```

We need to write our own clone function.
public Object clone() throws CloneNotSupportedException {
    X xob = null;
    xob = (X) super.clone();
    // now clone the array
    xob.arr = (int[]) arr.clone();
    return xob;
}

public static void main(String[] args) throws Exception {
    X xobj = new X();
    X xobjClone = (X) xobj.clone();
    System.out.println(xobj); // 0 4 5 2 5
    System.out.println(xobjClone); // 0 4 5 2 5
    xobj.arr[0] = 1000;
    System.out.println(xobj); // 1000 4 5 2 5
    System.out.println(xobjClone); // 0 4 5 2 5
X is as before

public Object clone() throws CloneNotSupportedException {
    X xob = null;
    xob = (X) super.clone();
    // now clone the array
    xob.arr = (int[]) arr.clone();
    return xob;
}
...

clone the object

header
arr
xobj

header
arr
xob

0 1 2 3 4
X is as before
public Object clone( ) throws CloneNotSupportedException {
    X xob = null;
xob = (X) super.clone( );
    // now clone the array
    xob.arr = (int[ ]) arr.clone( );
    return xob;
}
What about arrays of references?

header
arr
\( \times \text{obj} \)

\( \text{obj}_1 \)  \( \text{obj}_2 \)  \( \text{obj}_n \)
An example of cloning with arrays of objects

public class I implements Cloneable {

    int i;
    public I() {i = 0;}
    public I(int i) {this.i = i;}
    public void print() {System.out.println("i: "+i);
    public Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
}

I is a class that holds an integer
public class L implements Cloneable {

    I arryl[ ];

    public L( ) {
        arryl = new I[5];
        for (int i = 0; i < 5; i++) {
            arryl[i] = new I(i);
        }
    }

    void print(String s) {
        System.out.println("Printing L object "+s+":");
        for (int i = 0; i < arryl.length; i++)
            arryl[i].print( );
    }

    public void setElement(int i, int v) {
        arryl[i] = new I(v);
    }

    public Object clone( ) throws CloneNotSupportedException { // see next slide }
}

L is a class that has a reference to an array of I objects.
The clone method

```java
public Object clone( ) throws CloneNotSupportedException {
    L lClone = (L) super.clone( );

    // lClone.arrail = arryI.clone( ); WRONG by itself and
    // inefficient in the best case (recopies contents of arryI which
    // are then written over
    I arryClone[ ] = new I[arryI.length];

    for (int i = 0; i < arryI.length; i++)
        arryClone[i] = (I) arryI[i].clone( );

    lClone.arrail = arryClone;
    return lClone;
}
```
Let's see what this does pictorially

```java
public Object clone() throws CloneNotSupportedException {
    L lClone = (L) super.clone();
    // lClone.arryl = arryI.clone(); Why is this wrong?
    I arryClone[] = new I[arryl.length];
    for (int i = 0; i < arryI.length; i++)
        arryClone[i] = (I) arryI[i].clone();
    lClone.arryl = arryClone;
    return lClone;
}
```
Let's see what this does pictorially

```java
public Object clone() throws CloneNotSupportedException {
    L lClone = (L) super.clone();
    // lClone.arryl = arryI.clone(); Why is this wrong?
    I arryClone[] = new I[arryI.length];
    for (int i = 0; i < arryI.length; i++)
        arryClone[i] = (I) arryI[i].clone();
    lClone.arryI = arryClone;
    return lClone;
}
```
Let's see what this does graphically

public Object clone( ) throws CloneNotSupportedException {
    L lClone = (L) super.clone( );
    // IClone.arryI = arryI.clone( ); Why is this wrong?
    I arryClone[ ] = new I[arryI.length];
    for (int i = 0; i < arryI.length; i++)
        arryClone[i] = (I) arryI[i].clone( );
    IClone.arryI = arryClone;
    return lClone;
}

Creates storage for arryI, copies data from this.arryI, but doesn't clone the actual objects. There is a wasted copy.
public Object clone( ) throws CloneNotSupportedException {
    L IClone = (L) super.clone( );
    // IClone.arryl = arryI.clone( ); WRONG
    IarrayClone[ ] = new I[arryl.length];
    for (int i = 0; i < arryI.length; i++)
        IarrayClone[i] = (I) arryI[i].clone( );
    IClone.arryI = IarrayClone;
    return IClone;
}

Creates storage for IarrayClone.
Let's see what this does graphically

```java
public Object clone( ) throws CloneNotSupportedException {
    L IClone = (L) super.clone( );
    // IClone.arryI = arryI.clone( );;WRONG
    IarryClone[ ] = new I[arryl.length];
    for (int i = 0; i < arryI.length; i++)
        IarryClone[i] = (I) arryI[i].clone( );
    IClone.arryl = arryClone;
    return IClone;
}
```

clone the objects pointed to by the elements of larryClone
Let's see what this does graphically

```java
public Object clone() throws CloneNotSupportedException {
    L lClone = (L) super.clone();
    // IClone.arryl = arryI.clone(); WRONG
    IarryClone[ ] = new I[arryI.length];
    for (int i = 0; i < arryI.length; i++)
        IarryClone[i] = (I) arryI[i].clone();
    IClone.arryl = arryClone;
    return IClone;
}
```

cloned the objects pointed to by the elements of IarryClone
Let's see what this does graphically

```java
public Object clone() throws CloneNotSupportedException {
    L lClone = (L) super.clone();
    // lClone.arryI = arryI.clone(); // WRONG
    IarryClone[] = new I[arryI.length];
    for (int i = 0; i < arryI.length; i++)
        larryClone[i] = (I) arryI[i].clone();
    lClone.arryI = arryClone;
    return lClone;
}
```

cloning the objects pointed to by the elements of `larryClone`
Let's see what this does graphically

```java
class L {
    public Object clone() throws CloneNotSupportedException {
        L lClone = (L) super.clone();
        // lClone.arryI = arryI.clone(); // WRONG
        IarryClone[] = new I[arryI.length];
        for (int i = 0; i < arryI.length; i++)
            larryClone[i] = (I) arryI[i].clone();
        lClone.arryI = arryClone;
        return lClone;
    }
}

clone the objects pointed to by the elements of larryClone
```
Let's see what this does graphically

```java
public Object clone() throws CloneNotSupportedException {
    L IClone = (L) super.clone();
    // IClone.arryI = arryI.clone(); // WRONG
    IarryClone[] = new I[arryI.length];
    for (int i = 0; i < arryI.length; i++)
        IarryClone[i] = (I) arryI[i].clone();
    IClone.arryI = arryClone;
    return IClone;
}
```

clone the objects pointed to by the elements of larryClone
Let's see what this does graphically

```java
public Object clone( ) throws CloneNotSupportedException {
    L lClone = (L) super.clone( );
    // lClone.arryl = arryI.clone( ); //WRONG
    IarryClone[ ] = new I[arryI.length];
    for (int i = 0; i < arryI.length; i++)
        larryClone[i] = (I) arryI[i].clone( );
    lClone.arryl = arryClone;
    return lClone;
}
```

Make IClone.arryl reference larryClone
Let's see what this does graphically

```java
public Object clone( ) throws CloneNotSupportedException {
    L lClone = (L) super.clone( );
    // lClone.arryl = arryI.clone( );;WRONG
    IarryClone[ ] = new I[arryI.length];
    for (int i = 0; i < arryI.length; i++)
        IarryClone[i] = (I) arryI[i].clone( );
    lClone.arryl = arryClone;
    return lClone;
}
```

Returned the cloned object
In summary . . .

```java
public Object clone() throws CloneNotSupportedException {
    L lClone = (L) super.clone();
    // lClone.arryl = arryl.clone(); WRONG!
    l arryClone[ ] = new l[arryl.length];
    for (int i = 0; i < arryl.length; i++)
        arryClone[i] = (l) arryl[i].clone();
    lClone.arryl = arryClone;
    return lClone;
}
```

This clones the fields of the object, but not what those fields point to.
In summary . . .

```java
public Object clone() throws CloneNotSupportedException {
    L IClone = (L) super.clone();
    // IClone.arrayI = arrayI.clone(); WRONG!
    I arrayClone[] = new I[arrayI.length];
    for (int i = 0; i < arrayI.length; i++)
        arrayClone[i] = (I) arrayI[i].clone();
    IClone.arrayI = arrayClone;
    return IClone;
}
```

Create a new array of I objects for the cloned L object.
In summary . . .

public Object clone() throws CloneNotSupportedException {
    L lClone = (L) super.clone();
    // lClone.arryl = arryl.clone(); WRONG!
    I arryClone[ ] = new I[arryl.length];
    for (int i = 0; i < arryl.length; i++)
        arryClone[i] = (I) arryl[i].clone();
    lClone.arryl = arryClone;
    return lClone;
}
public class Test {

    public static void main(String[] args) throws Exception {
        L lobj = new L();
        lobj.print("lobj");
        L lobjCloned = (L) lobj.clone();
        lobjCloned.print("lobj Cloned");
        lobjCloned.setElement(2, 500);
        lobj.print("lobj");
        lobjCloned.print("lobjCloned");
    }
}

Driver code for the example
public class Test {
    public static void main(String[] args)
        throws Exception {
        L lobj = new L();
        lobj.print("lobj");
        L lobjCloned = (L) lobj.clone();
        lobjCloned.print("lobj Cloned");
        lobjCloned.setElement(2, 500);
        lobj.print("lobj");
        lobjCloned.print("lobjCloned");
    }
}
public class Test {
    public static void main(String[] args) throws Exception {
        L lobj = new L();
        lobj.print("lobj");
        L lobjCloned = (L) lobj.clone();
        lobjCloned.print("lobj Cloned");
        lobjCloned.setElement(2, 500);
        lobj.print("lobj");
        lobjCloned.print("lobjCloned");
    }
}

---

Printing L object lobj:
i: 0
i: 1
i: 2
i: 3
i: 4

Printing L object lobjCloned:
i: 0
i: 1
i: 500
i: 3
i: 4
Change L.java's clone to ...

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

Printing L object lobj:

i: 0
i: 1
i: 2
i: 3
i: 4

Printing L object lobj Cloned:

i: 0
i: 1
i: 2
i: 3
i: 4

lobjCloned.arryl[2]=500

Printing L object lobj:

i: 0
i: 1
i: 500
i: 3
i: 4

Printing L object lobjCloned:

i: 0
i: 1
i: 500
i: 3
i: 4
This comes from not implementing a public clone method in the class being cloned.
A short discussion of aliasing

• Consider the program

```java
Integer a = new Integer(57);
Integer b = new Float(58.0);
Object c = null;

... if (some condition) {
    c = a;
} else {
    c = b;
}
System.out.println(c);
```

• At the `println` what gets printed?
• It could be 57, or it could be 58.0.
• In general, there is not way for the programmer or a compiler to tell
• For a compiler, its equivalent to the *halting problem.*