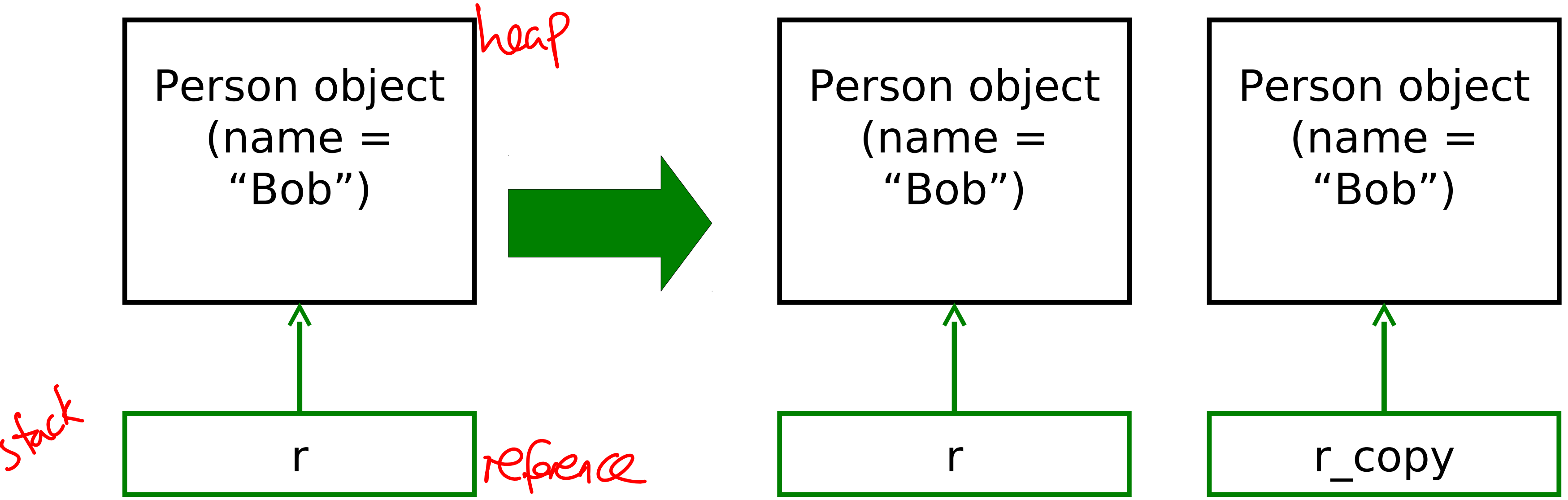
Section: Copying Java Objects

Cloning I

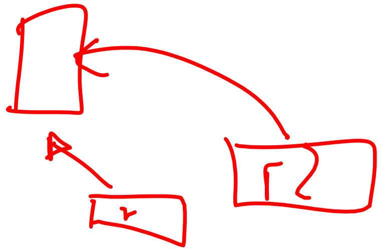
- Sometimes we really do want to copy an object



heap
Person object (name = "Bob")
stack
r
reference
Person object (name = "Bob")
r
Person object (name = "Bob")
r_copy

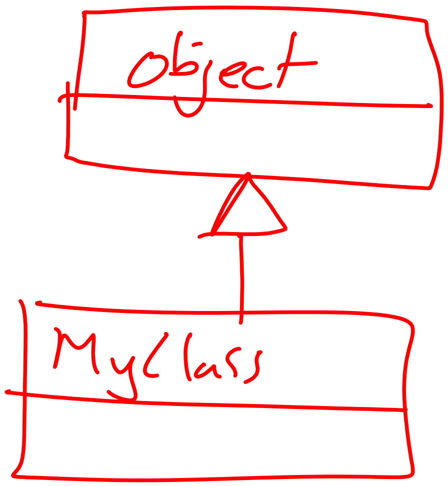
- Java calls this ***cloning***
- We need special support for it

r2 = r;



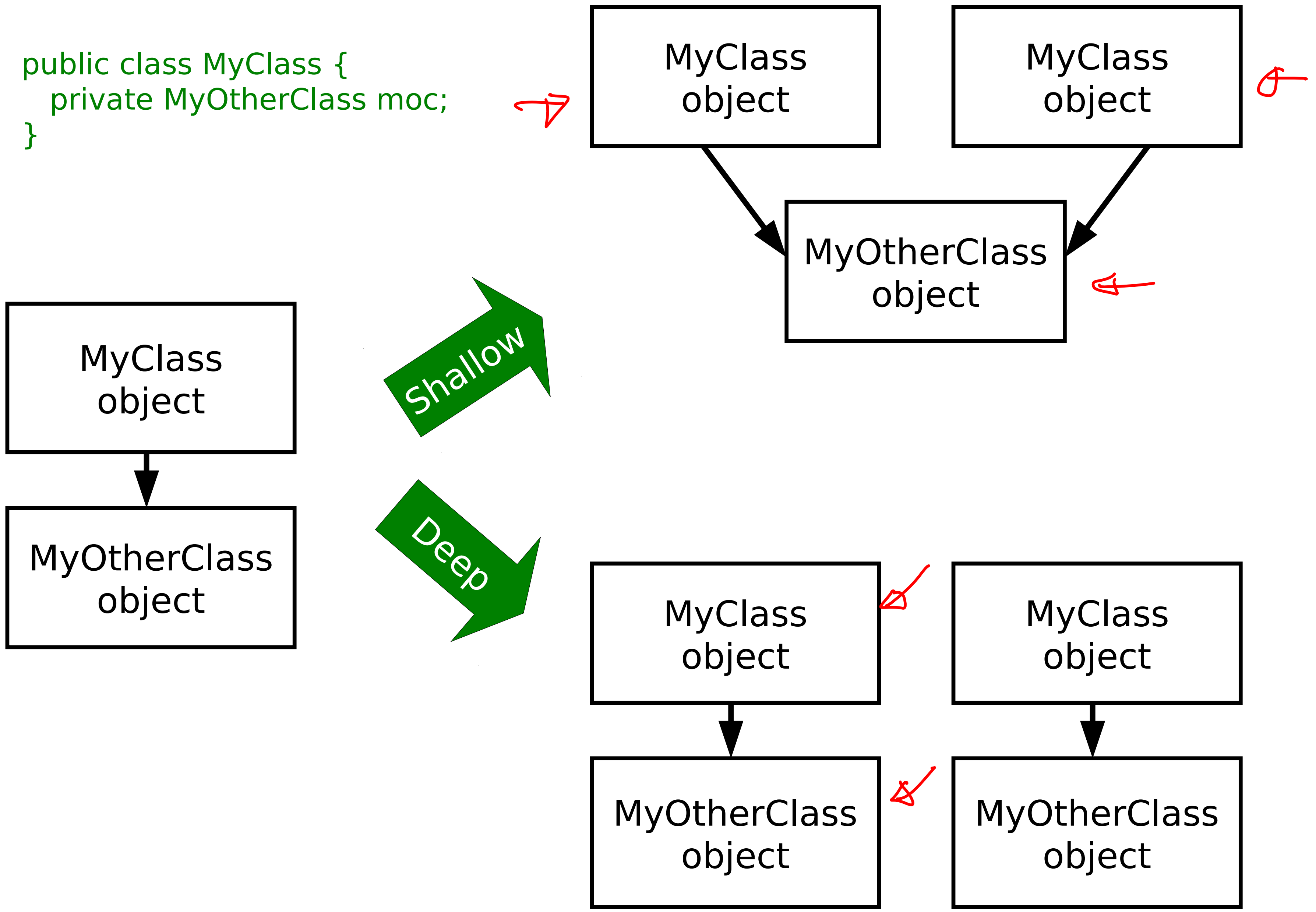
Cloning II

- Every class in Java ultimately inherits from the **Object** class
  - This class contains a clone() method so we just call this to clone an object, right?
  - This can go horribly wrong if our object contains reference types (objects, arrays, etc)



Shallow and Deep Copies

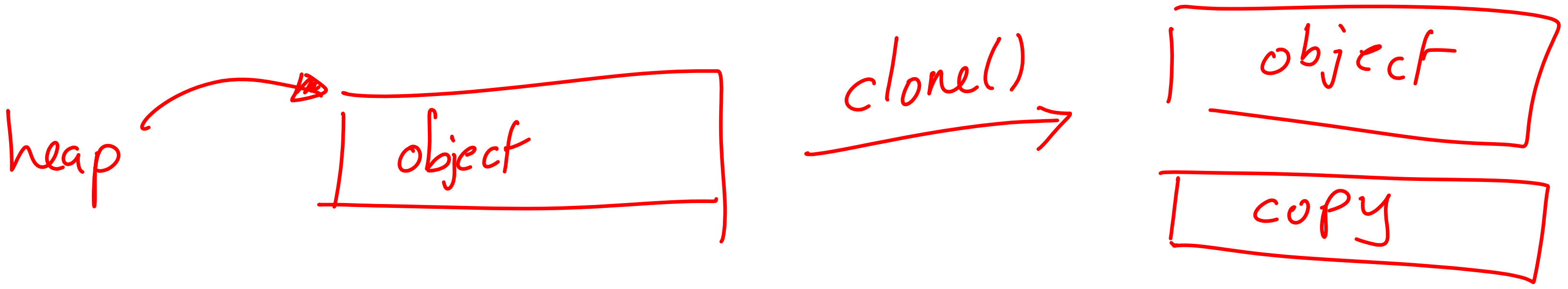
```
public class MyClass {
   private MyOtherClass moc;
}
```



Java Cloning

- So do you want shallow or deep?
  - The default implementation of clone() performs a **shallow** copy
  - But Java developers were worried that this might not be appropriate: they decided they wanted to know for <u>sure</u> that we'd thought about whether this was appropriate

- Java has a **Cloneable** interface
  - If you call clone on anything that doesn't extend this interface, it fails



Clone Example I

```
public class Velocity {
  public float vx;
  public float vy;
  public Velocity(float x, float y) {
      vx=x;
      vy=y;
  }
 };

 public class Vehicle {
  private int age;
  private Velocity vel;
  public Vehicle(int a, float vx, float vy) {
      age=a;
      vel = new Velocity(vx,vy);
  }
 };
```

want to clone

Clone Example II

```
public class Vehicle implements Cloneable {
  private int age;
  private Velocity vel;
  public Vehicle(int a, float vx, float vy) {
     age=a;
     vel = new Velocity(vx,vy);
  }

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

};
```

throws
CloneNotSupportedException

Shallow

Clone Example III

```
 public class Velocity implement Cloneable {
      ....
     public Object clone() {
         return super.clone();
     }
};


public class Vehicle implements Cloneable {
  private int age;
  private Velocity v;
  public Student(int a, float vx, float vy) {
      age=a;
      vel = new Velocity(vx,vy);
  }

  public Object clone() {
      Vehicle cloned = (Vehicle) super.clone();
      cloned.vel = (Velocity)vel.clone();
      return cloned;
  }

};
```

Making something Cloneable ← Java specific!

1. Implement Cloneable interface
2. Make clone() accessible (public)
3. Call super.clone() ← shallow copy
4. (Recursively) clone all objects in your class

Marker Interfaces

- If you look at what's in the Cloneable interface, you'll find it's empty!! What's going on?
- Well, the clone() method is already inherited from Object so it doesn't need to specify it
- This is an example of a **Marker Interface**
    - A marker interface is an empty interface that is used to label classes
    - This approach is found occasionally in the Java libraries

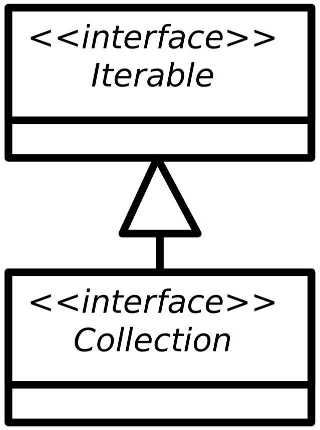
Section: The Java Class Libraries

Java Class Library

- Java the platform contains around 4,000 classes/interfaces
  - Data Structures
  - Networking, Files
  - Graphical User Interfaces
  - Security and Encryption
  - Image Processing
  - Multimedia authoring/playback
  - And more…

- All neatly(ish) arranged into packages (see API docs)

Java's Collections Framework

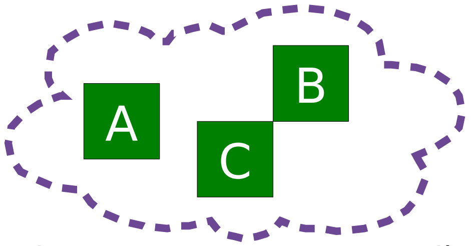


- Important chunk of the class library
- A collection is some sort of grouping of things (objects)
- Usually when we have some grouping we want to go through it ("***iterate*** over it")

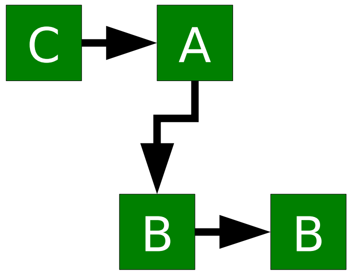
- The Collections framework has two main interfaces: Iterable and Collections. They define a set of operations that all classes in the Collections framework support
- add(Object o), clear(), isEmpty(), etc.

Major Collections Interfaces I

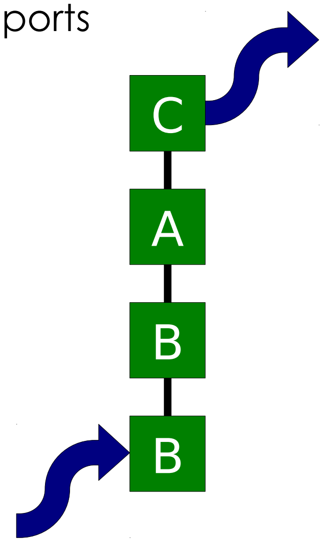
- <<interface>> Set
  - Like a mathematical set in DM 1
  - A collection of elements with no duplicates
  - Various concrete classes like TreeSet (which keeps the set elements sorted)



- <<interface>> List
  - An ordered collection of elements that may contain duplicates
  - ArrayList, Vector, LinkedList, etc.



- <<interface>> Queue
  - An ordered collection of elements that may contain duplicates and supports removal of elements from the head of the queue
  - PriorityQueue, LinkedLIst, etc.



Major Collections Interfaces II

- <<interface>> Map
  - Like relations in DM 1, or dictionaries in ML
  - Maps key objects to value objects
  - Keys must be unique
  - Values can be duplicated and (sometimes) null.