Passing and Returning Objects from Methods

Recall the call by value behavior we get when we send a primitive data type in as a parameter:

```
public class Foo
{
      public static void doesntChange(int num1)
      {
            int y = 3;              // Local variable, used later
            System.out.println(num1);        // Prints 1
            num1 = 5;
            System.out.println(num1);        // Prints 5
            return;
      }


      public static void main(String[] args)
      {
            int val = 1;
            doesntChange(val);
            System.out.println("Back in main:" + val); // Prints 1
      }
}
```

The output for this program is:

```
1
5
Back in main: 1
```

We send in a copy of val into the method. Num1 is a separate variable with the value 1.

Side Q: How would we invoke the method if it is not static?

(In class – describe the stack again and how the value 1 is copied into the space allocated for num1).

We get different behavior if we pass an Object. Consider the Foo class:

```
public class Foo
{
      public int val;  // Public for simplicity
      public Foo()
      {
```

```
                  val = 0;
            }
            public Foo(int val)
            {
                  this.val = val;
            }
      }
```

Now we pass a Foo object to a method:

```
      public class Test{

            public static void methodCall(Foo obj)
            {
                  System.out.println(obj.val); // Outputs 3
                  obj.val = 10;
                  System.out.println(obj.val); // Outputs 10
            }
            public static void main(String[] args)
            {
                  Foo f = new Foo(3);
                  methodCall(f);
                  System.out.println(f.val); // Outputs 10
            }
      }
```

This outputs 10 back in main! The contents of the object are changed! Passing an object to a method changes the contents of the object. This is because the object is passed by reference.

(In class – show stack and how pass by reference works to change the original object).

Note that arrays are also passed by reference. If a method changes an array then it will be changed back in the calling code. This is because arrays are objects.

A method can return only one value. What if you want to return more than one thing? You can have a method return an object with the items to send back. Here is an example where a method returns a name and ID wrapped inside a Person object:

```
      public class Person
      {
            public String name;  // Public for simplicity
            public int ID;
```

```
	public Person()
	{
		name = "";
		ID = 0;
	}
	public Person(String n, int i)
	{
		name = n;
		id = i;
	}
}

public class Test
{

	public static Person getPerson()
	{
		Scanner keyboard = new Scanner(System.in);
		System.out.println("Name?");
		String name = keyboard.nextLine();
		System.out.println("ID?");
		int id = keyboard.nextInt();
		return new Person(name,id);
	}


	public static void main(String[] args)
	{
		Person someone = getPerson();
		System.out.println(someone.name + " " +
someone.ID);
	}
}
```

The new person entered is returned back to main where it can be used.