CS 2100: Data Structures & Algorithms 1

Methods and Parameters

Dr. Nada Basit // basit@virginia.edu

Spring 2022

Friendly Reminders

- Masks are **required** at all times during class (University Policy)
- If you forget your mask (or mask is lost/broken), I have a few available
  - Just come up to me at the start of class and ask!
- No eating or drinking in the classroom, please
- Our lectures will be **recorded** (see Collab) – please allow 24-48 hrs to post
- If you feel **unwell**, or think you are, please stay home
  - *We will work with you!*
  - At home: eye mask instead! Get some rest ☺

Reminder: Java is Object Oriented

- Everything in Java is an **Object**
  - Objects have **state** and **behavior**
    - **State**: properties (*fields*, *variables*) of an object
    - **Behavior**: **methods** (*functions*) of an object
  - Example: A Cat object could be asked:
    - *How many legs do you have?*
    - *What is your name?*
    - *Take a nap!*
    - *Play with yarn!*

In object-oriented programming a "function" is called a "**method**" (similar meaning)

Basics of Methods

Brief Overview of Methods

- In Java, any code you write MUST be **within a class**
  - More on classes, what they are, etc. in an up-coming lecture!
- Java methods behave more or less like methods in other languages:
  - Can take parameters
  - Have return values
- Methods in Java are associated with a single class definition

Reminder: Layout of the Class "BasicFunctions"

- There are no nested methods in a Java program. Each method sits inside the class. Order doesn't matter!

```
public class BasicMethods {

   public static void main(String[] args) {
      // declare and initialize two int variables:
      int a = 5;
      int b = 7;
      System.out.println("The sum is: " + add(a,b) ); // call add() method
   }

   public static int add(int x, int y) {
      return x + y; // add the two numbers and return the result
   }
}
```

Methods

- **Methods** are functions written inside of a class

```
public class Cat {
      int numLegs = 4;              // field (variable)
      String name = “Ginger”;       // field (variable)
      public void takeNap(arg) { // method header
            …stuff…
      }
      …
```

- **Method headers** are comprised of the access specifier, return type, name and argument(s)

A Simple Method

```
import java.util.Scanner;
public class MethodExample {
    /**
      * This is a method
      * takes in two parameters and returns the larger
      */
    public static int getMax(int num1, int num2){
      return (num1 >= num2 ? num1 : num2);
    }

    public static void main(String args[]) {
        Scanner in = new Scanner(System.in);
        int x1 = in.nextInt();
        int x2 = in.nextInt();

        System.out.println("The bigger number is " + getMax(x1, x2));
    }
}
```

Some More Notes On Methods

- Scope
  - **Public**: Anyone can invoke this method
  - **Private**: Method can only be invoked from within this class
  - **Protected**: Method can be invoked by inheriting classes and ones in the same package
  - *More on this later...*
- Static
  - Basically means the class only has **one** shared instance of this method *(Again, more on this later...)*
- Return value
  - Can be `void` if no return necessary, otherwise methods MUST contain a **`return`** statement
- Parameters
  - Can pass as many parameters as you want, but **must declare the types!**

Method Headers / Overloading Methods

- Methods may have the **SAME NAME** within a class, but not the same exact header!

- Methods with the same name **must** have different arguments
  - Java compiler can distinguish methods by arguments
    - **Number** of arguments
    - **Data type** of arguments
    - **Position (order)** of arguments
  - This technique is called method **OVERLOADING**

Method Overloading Example

```
// 1: Overloaded sum(). This sum takes two int parameters
public static int sum(int x, int y) {
    return (x + y);
}

// 2: Overloaded sum(). This sum takes three int parameters
public static int sum(int x, int y, int z) {
    return (x + y + z);
}

// 3: Overloaded sum(). This sum takes two double parameters
public static double sum(double x, double y) {
    return (x + y);
}
```

Which method is called? `System.out.println( sum(4, 18) );`

Parameter Passing

Method Arguments

Reminder: Formal vs. Actual Parameters

```
public class SimpleMethod {

  public static void main(String[] args) {
    int a = 5; // declare and initialize two int variables: a and b
    int b = 7;
    System.out.println("The sum is: " + sum(a,b) ); // actual parameters
  }

  public static int sum(int x, int y) { // formal parameters
    return x + y; // add the two numbers and return the result
  }
}
```

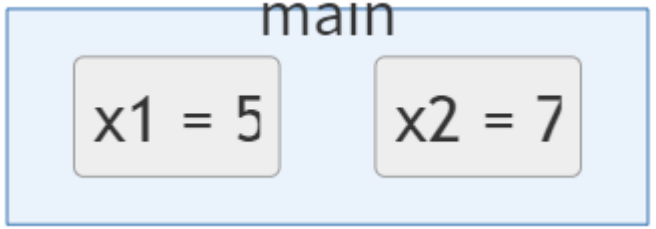
Formal parameters (e.g. x and y) must have a declared type (e.g. **int**) – the variable(s) the method uses
The **sum()** method takes as input two integer variables, invoked using 'a' and 'b', called **actual parameters**

Method Arguments

- Java is ***pass-by-value*** always 
  - Done for safety as well as efficiency.
  - The value on the stack is copied into the new method's parameter.
  - This means that a **copy** of the **actual parameter** is made into the formal parameter, and the method operates on the **copy** of that.
- Can produce surprising results – *be mindful!*
  - **Primitive** variables have their **values** copied.
  - **Reference** variables have their "**pointers**" (*references*) copied: both will point to the same object!

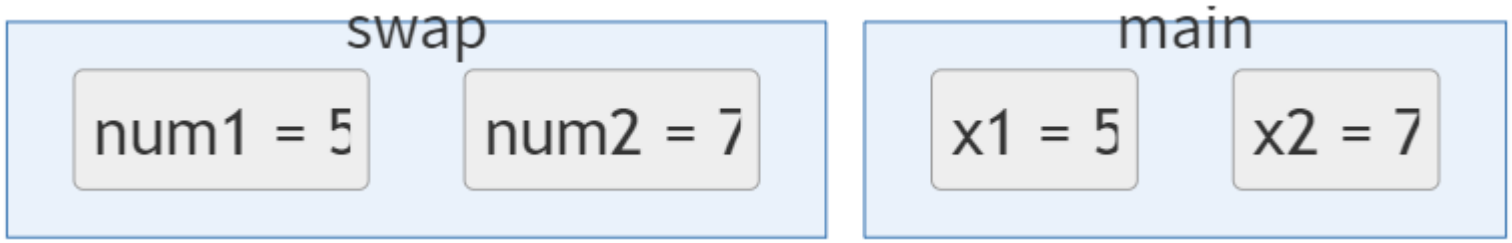
Parameter Passing Examples: SWAP

```
public static void swap(int num1, int num2){
   int temp = num1;
   num1 = num2;
   num2 = temp;
}
public static void main(String args[]) {
   int x1 = 5; int x2 = 7;
   swap(x1,x2); // <== Code is here
   System.out.println("After: x1: " + x1 + "  x2: " + x2);
}
```



Parameter Passing Examples: SWAP

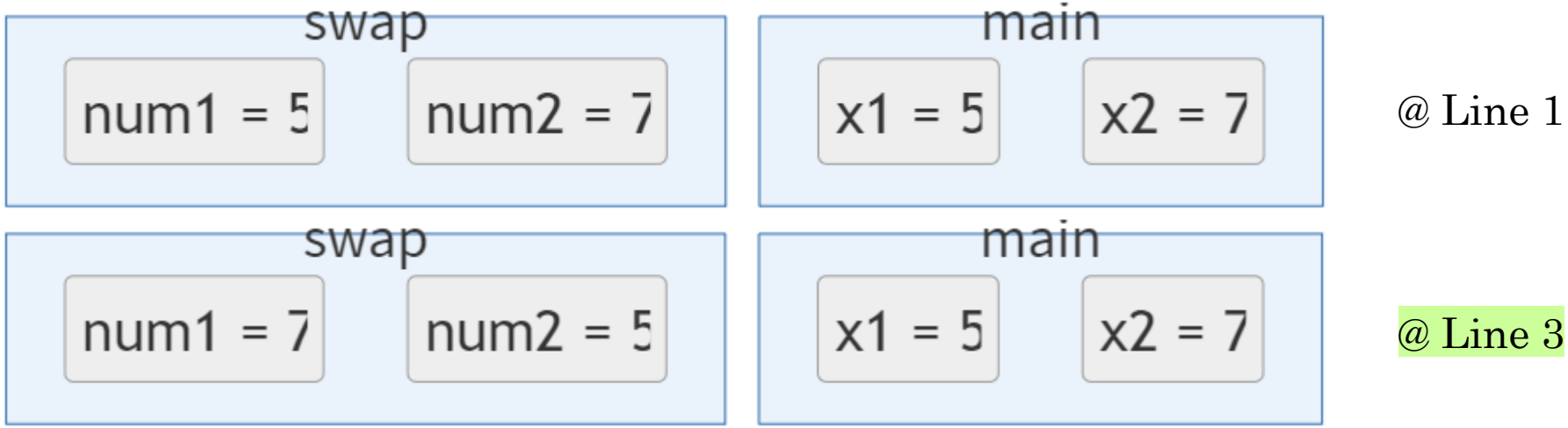
```
public static void swap(int num1, int num2){
   int temp = num1; // <== Code is here
   num1 = num2;
   num2 = temp;
}
public static void main(String args[]) {
   int x1 = 5; int x2 = 7;
   swap(x1,x2);
   System.out.println("After: x1: " + x1 + "  x2: " + x2);
}
```



Parameter Passing Examples: SWAP

```
public static void swap(int num1, int num2){
   int temp = num1; // Line 1
   num1 = num2;
   num2 = temp; // <== Code is here (Line 3)
}
public static void main(String args[]) {
   int x1 = 5; int x2 = 7;
   swap(x1,x2);
   System.out.println("After: x1: " + x1 + "  x2: " + x2);
}
```

The swap **does** happen ***locally*** inside of the swap() method. num1 holds the value of num2, and num2 holds the value of num1.

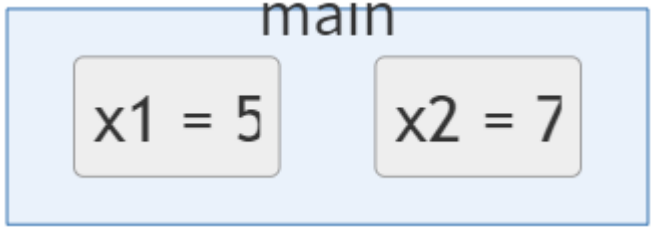


@ Line 1

@ Line 3

Parameter Passing Examples: SWAP

```
public static void swap(int num1, int num2){
   int temp = num1;
   num1 = num2;
   num2 = temp;
}
public static void main(String args[]) {
   int x1 = 5; int x2 = 7;
   swap(x1,x2); // <== Code RETURNS here
   System.out.println("After: x1: " + x1 + "  x2: " + x2);
}
```



Original x1 and x2 remained unchanged!

Output:

```
After: x1: 5  x2: 7
```

Explanation:
We made a copy of the values

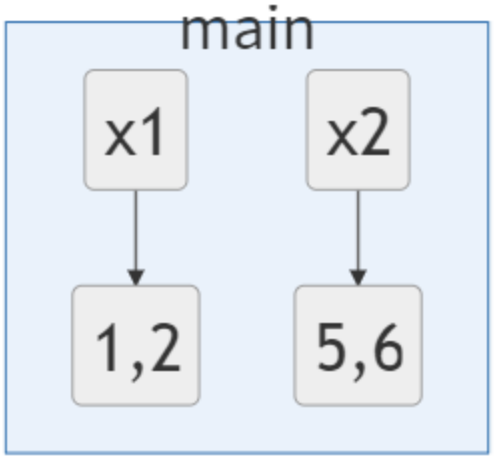
Swapping Objects (e.g., REFERENCES)

- **Be careful when passing a reference type by value!**
  - Will the following work?

```
public static void swap(Point p1, Point p2) {
   Point temp = p1;
   p1 = p2;
   p2 = temp;
}
public static void main(String args[]) {
   Point x1 = new Point(1,2); Point x2 = new Point(5,6);
   swap(x1,x2);
}
```

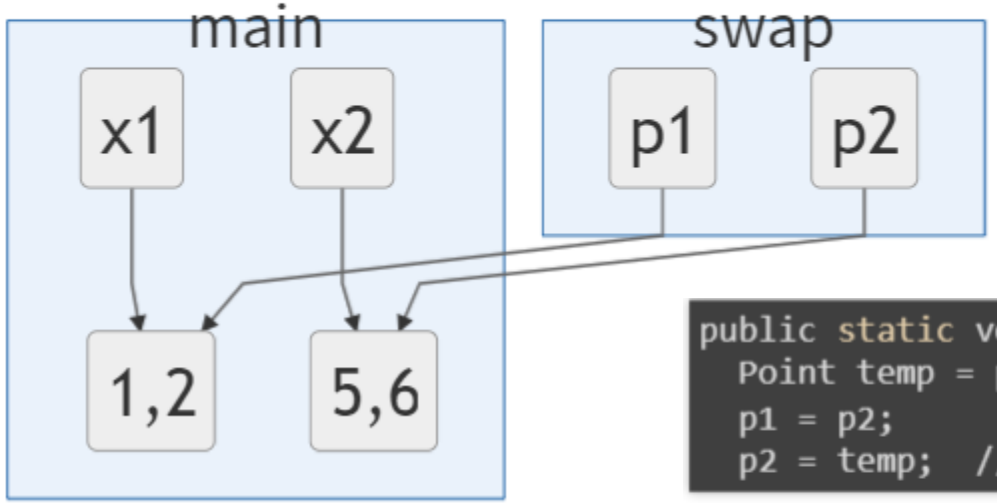
Swapping Objects (e.g., REFERENCES)

```
Point x1 = new Point(1,2); Point x2 = new Point(5,6);
swap(x1,x2);
```

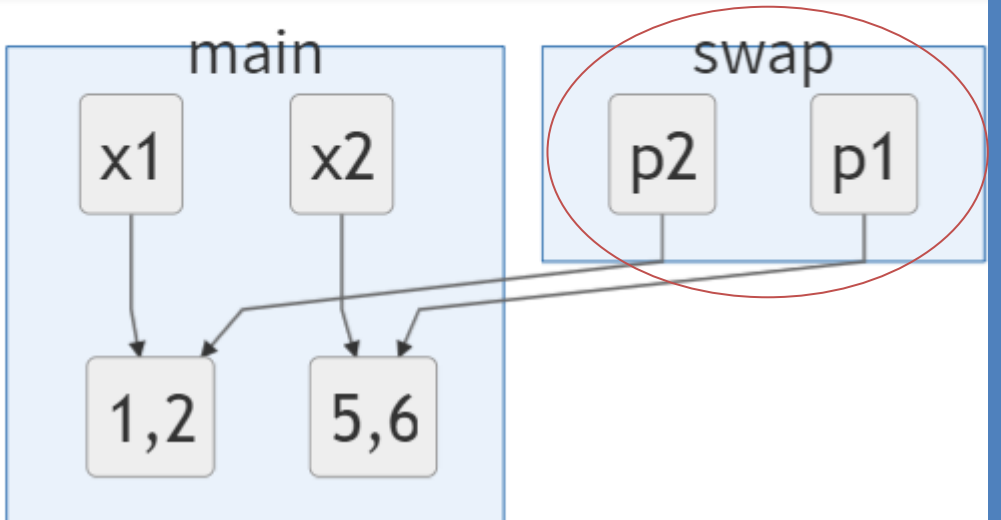


Swapping Objects (e.g., REFERENCES)

```
public static void swap(Point p1, Point p2){
  Point temp = p1;  //CODE IS HERE
  p1 = p2;
  p2 = temp;                                  }
```



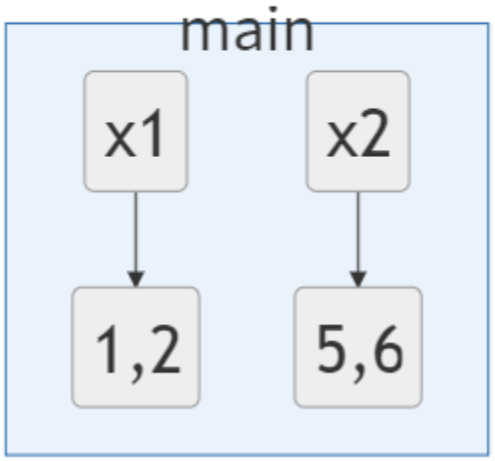

```
public static void swap(Point p1, Point p2){
  Point temp = p1;
  p1 = p2;
  p2 = temp;  //CODE IS HERE                  }
```



Swapping Objects (e.g., REFERENCES)

- At the end of it all, x1 still points to (1,2) and x2 still points to (5,6).
- Why? We made a copy of the pointers (references) p1 and p2 and swapped them.

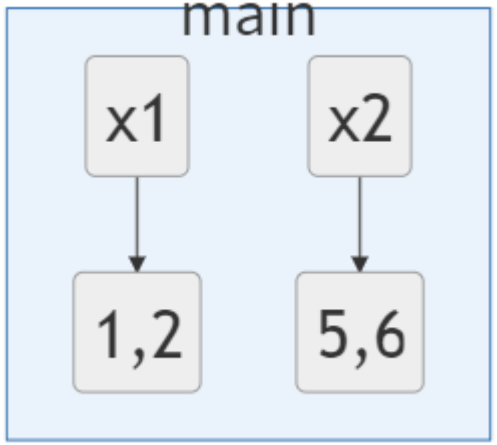
```
Point x1 = new Point(1,2); Point x2 = new Point(5,6);
swap(x1,x2); //Method returns, no change in x1 or x2
```



Another Example! Swapping Point Objects

```
public static void swap(Point p1, Point p2) {
    Point temp = (Point) p1.clone(); //Deep Copy
    p1.x = p2.x; p1.y = p2.y;
    p2.x = temp.x; p2.y = temp.y;
}

public static void main(String args[]) {
    Point x1 = new Point(1,2); Point x2 = new Point(5,6);
    swap(x1,x2);
}
```

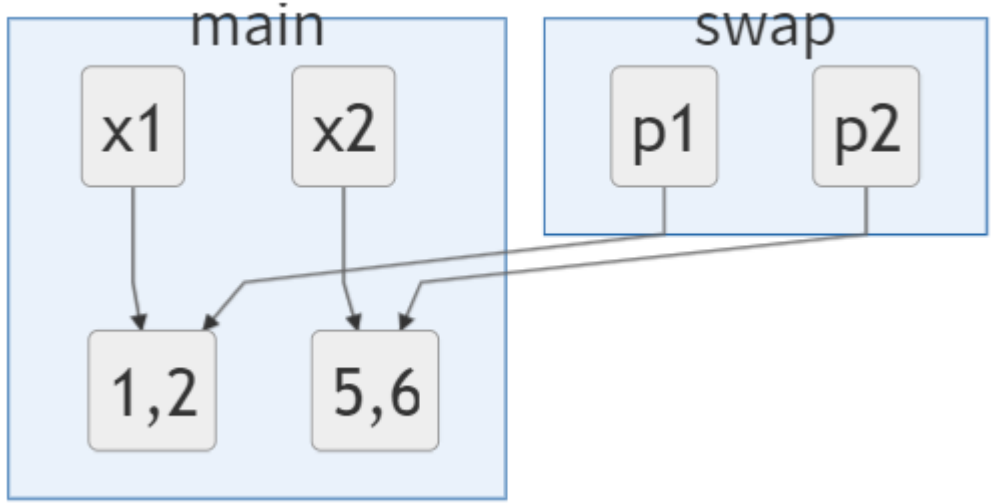


About to invoke **swap()** method

Another Example! Swapping Point Objects

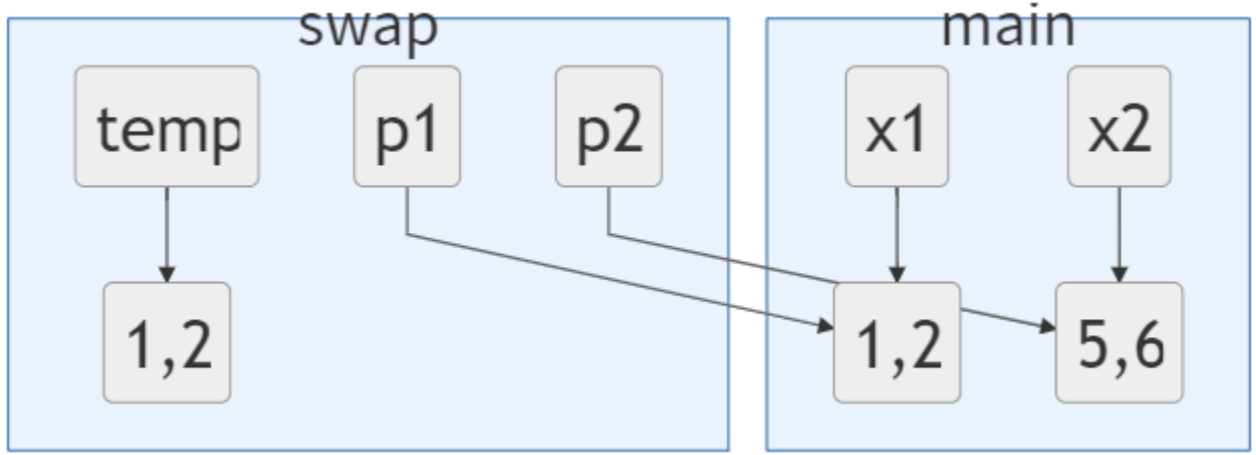
```
public static void swap(Point p1, Point p2) {
    Point temp = (Point) p1.clone(); //Deep Copy
    p1.x = p2.x; p1.y = p2.y;
    p2.x = temp.x; p2.y = temp.y;
}
```

Executing the **swap()** method



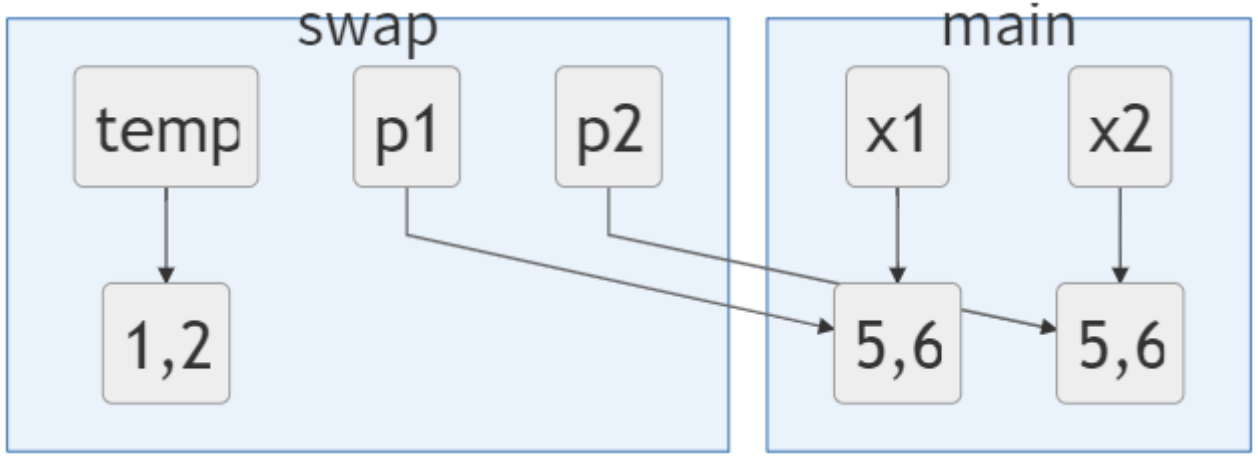
Another Example! Swapping Point Objects

```
public static void swap(Point p1, Point p2) {
    Point temp = (Point) p1.clone(); // <== This line executed
    p1.x = p2.x; p1.y = p2.y;
    p2.x = temp.x; p2.y = temp.y;
}
```



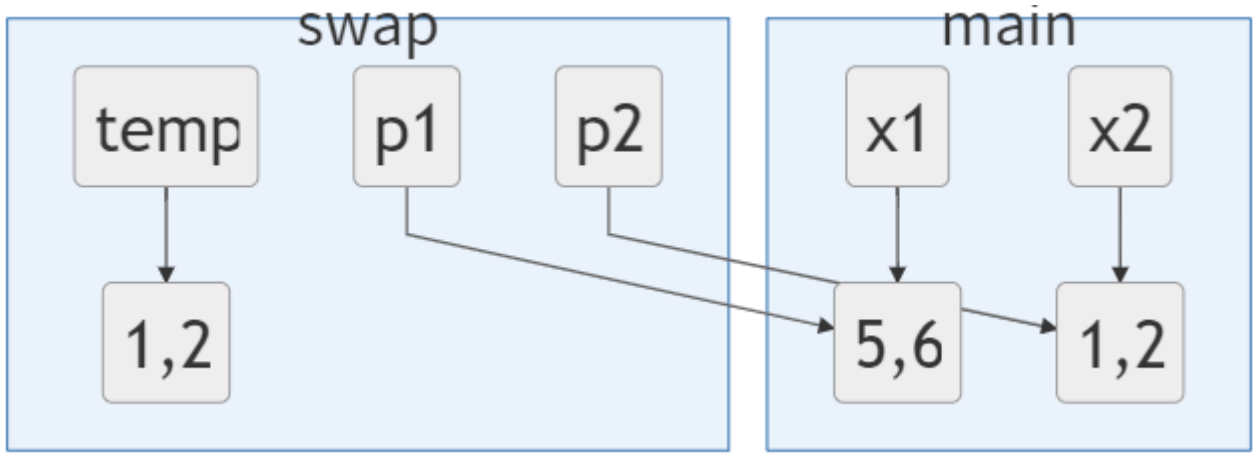
Another Example! Swapping Point Objects

```
public static void swap(Point p1, Point p2) {
    Point temp = (Point) p1.clone();
    p1.x = p2.x; p1.y = p2.y; // <== This line executed
    p2.x = temp.x; p2.y = temp.y;
}
```



Another Example! Swapping Point Objects

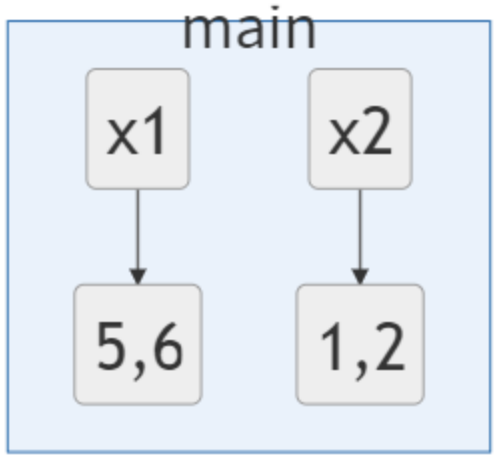
```
public static void swap(Point p1, Point p2) {
    Point temp = (Point) p1.clone();
    p1.x = p2.x; p1.y = p2.y;
    p2.x = temp.x; p2.y = temp.y; // <== This line executed
}
```



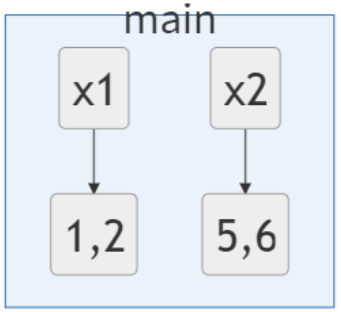
Another Example! Swapping Point Objects

- Now, swap() has returned.
- What does x1 and x2 in main() look like?

```
Point x1 = new Point(1,2); Point x2 = new Point(5,6);
swap(x1,x2); //Swap has returned
```



Used to be the following:



Parameters Summary

- When methods are called, a **copy** of the actual parameter is made into the formal parameter
  - For primitives, a copy of the data itself
  - For references, a copy of the memory address
    - Be careful with references
    - Reassigning the reference will not change the actual parameter, but altering what the reference points to will.
      This is an important distinction!

A Note About main() Method Testing...

Testing in main() method

- Many classes do not have or need a main() method
  - *A Java Class is perfectly complete without a main() method*
- The main() method is used when you want to do "stuff"
- You can utilize the main method to write tests for the other methods in the class
  → call methods, give inputs, observe outputs
  → use `System.out.println()` statements