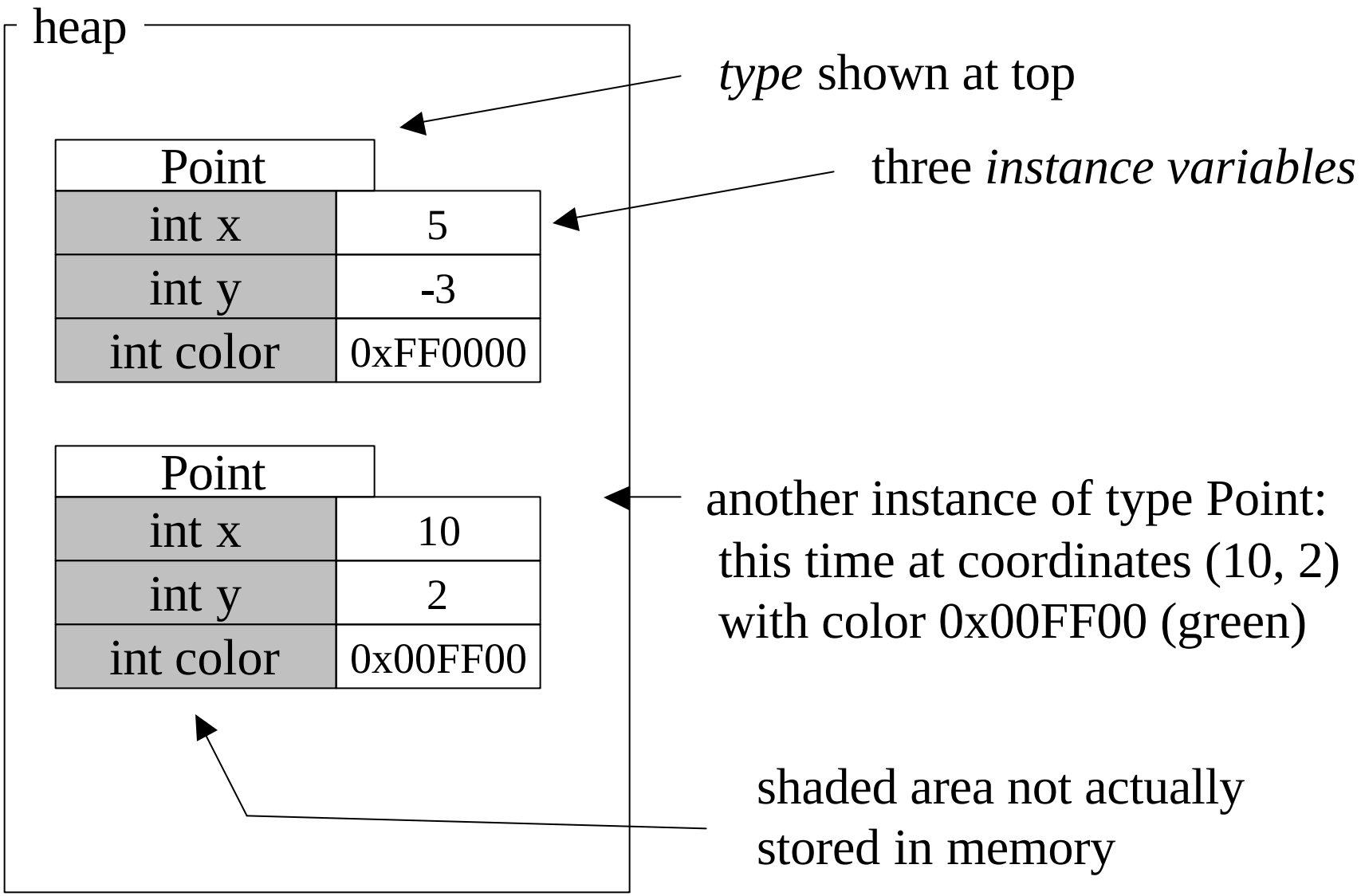
References and Objects

Weiss ch. 2

Objects

- Are *structured* regions of memory
  - Reside in heap
- Each object is an *instance* of a *type*
  - e.g. String, File, Point, arrayOfInteger, arrayOfString, etc.
  - “object” and “instance” are synonyms
- Each object has named contents
  - called *instance variables* or *member variables* or *fields*
  - Binding: Each name assigned one slot in the object’s memory region, as an offset from the start of the object
- Each object has associated methods
  - called *instance methods* or *member methods*
  - these are shared between all instances of a type

Example: A Point Instance

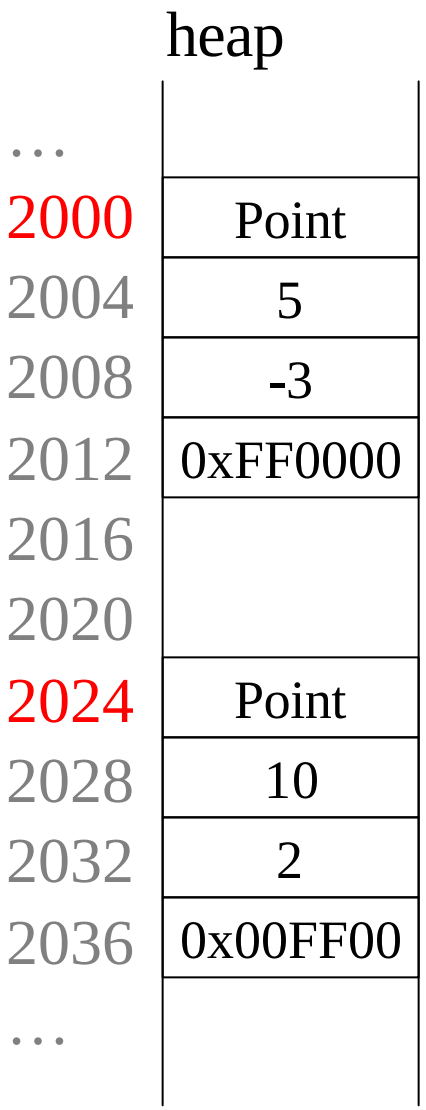


Objects

- What can we do with an object?
  - e.g., want to change the coordinates of “that second instance of Point that has color green”.
  - Need to *name* objects, so we can *refer* to them.

References

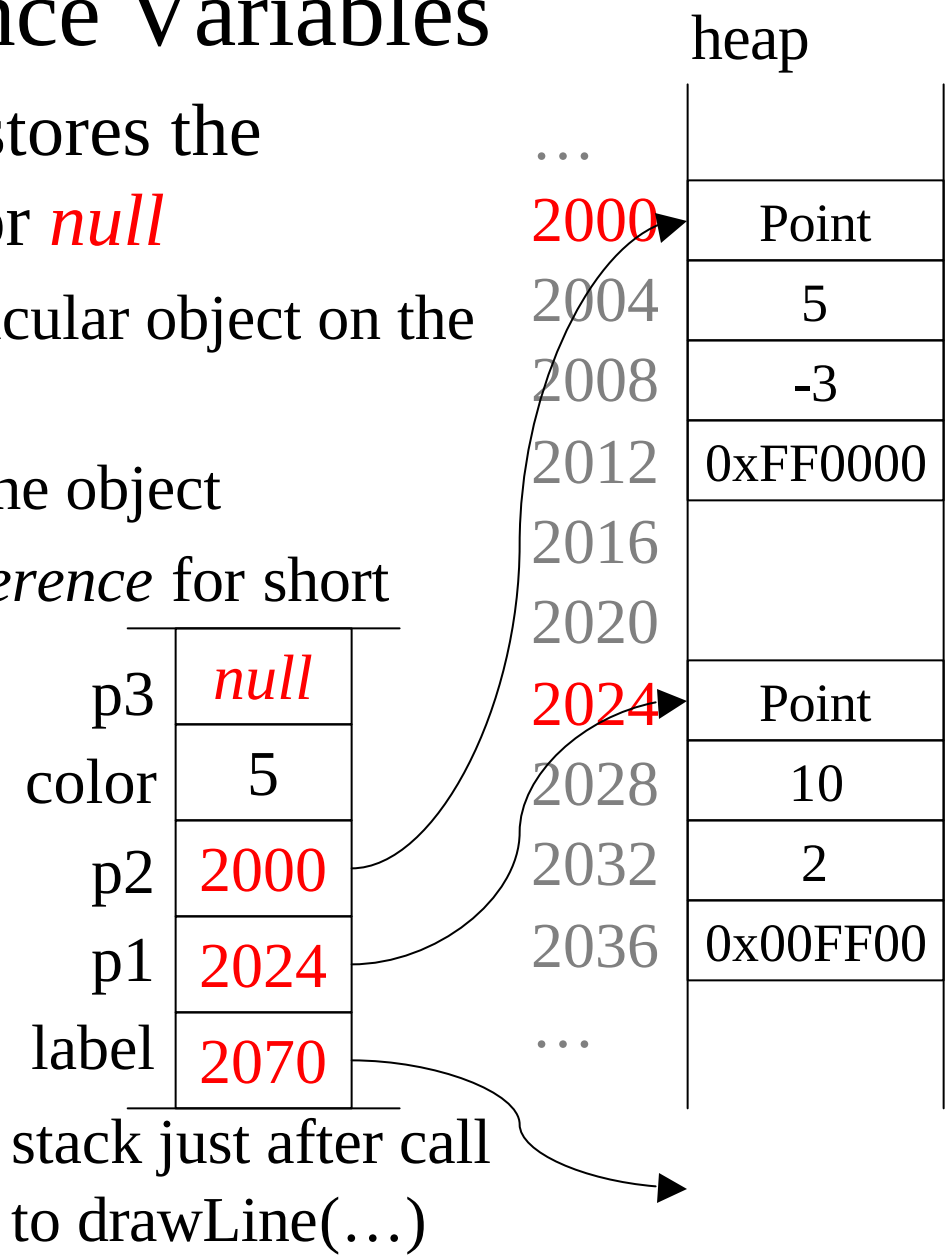
- Memory addresses
  - Each slot in memory has an address (number)
  - Physically and logically: each location in each chip of RAM is numbered consecutively, starting at 0 and counting upwards
- We can refer to an object by its *address* in the heap
  - In Java, you (the user) rarely get to see the actual address



Reference Variables

- A *reference variable* stores the address of an object, or *null*
  - Used to *refer* to a particular object on the heap, or to nothing
  - Think: a "pointer" to the object
  - Usually called just *reference* for short

```
static double drawLine(
  String label, // ref. to a String
  Point p1, // ref. to a Point
  Point p2, // ref. to a Point
  int color) {
    Point p3 = null;
  …
}
```



What can we do with a *reference*?

1. Assignment
   p1 = p2
2. Equality testing
   p1 == p2
   p1 != p2
3. Access the members of the object that it refers to ("points at"):
   p1.x
4. Type conversions (later)
5. instanceof operator (later)

these operate on the *reference variables* themselves, not the objects referred to

these operate on the *objects* that are pointed at

Constructing New Objects

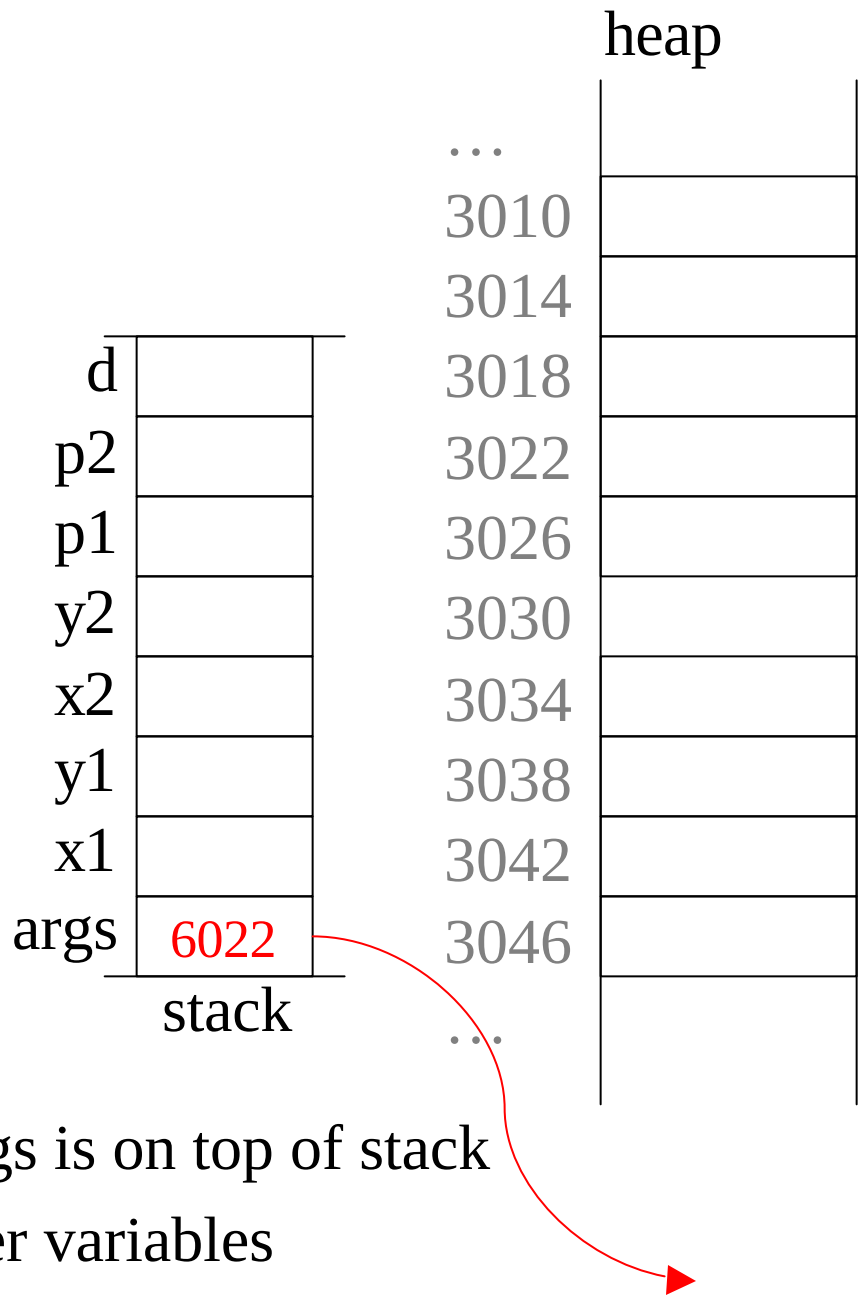
- new Point()
  - allocates space for a new object/instance of type Point on the heap
  - initializes its fields (to all zeros, in this case)
  - returns a reference to it
- Often specify parameters during creation:
  - new Point(5, -3) → initialize to x=5, y=-3, default color
  - new Point(10 2, RED) → initialize to x=10, y=2, color=RED

A Very Detailed Example

```
public static void main(String args[ ]) {
  int x1 = Integer.parseInt(args[0]);
  int y1 = Integer.parseInt(args[1]);
  int x2 = Integer.parseInt(args[2]);
  int y2 = Integer.parseInt(args[3]);
  Point p1 = new Point(x1, y1);
  Point p2 = new Point(x2, y2);
  double d = distance(p1, p2);
  if (d < 10) p1.color = RED;
  else p1.color = GREEN;
  System.out.println("Distance from "+p1+" to "+p2+" is "+d);
}

static double distance(Point a, Point b) {
  int dx = a.x - b.x;
  int dy = a.y - b.y;
  return Math.sqrt(dx*dx + dy*dy);
}
```

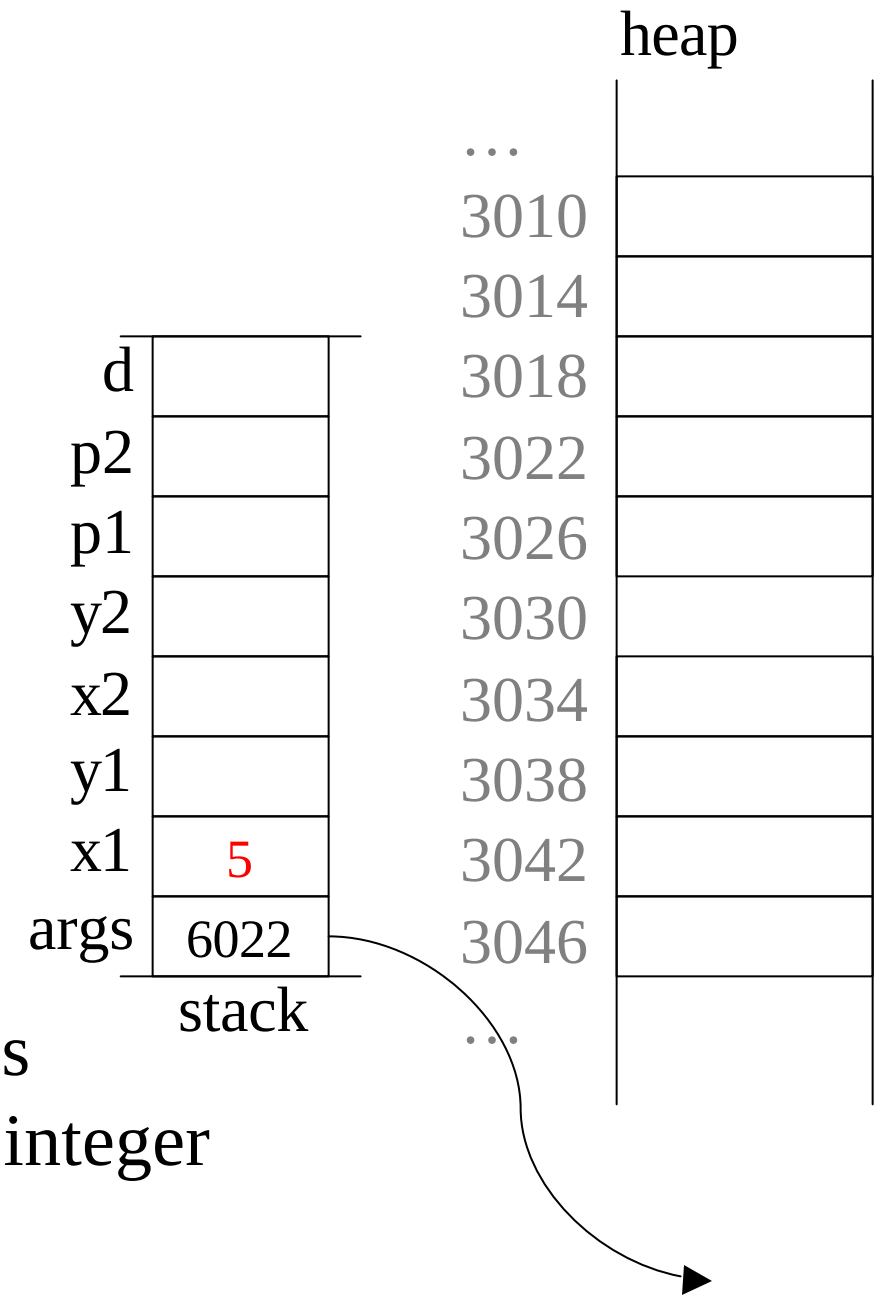
```
Welcome to DrJava.
> java Distance 5 -3 10 2
Distance from (5, -3, 0xff0000) to (10, 2, 0x0) is 7.071067811865475
```

```
public static void main(String args[ ]) {
  int x1 = Integer.parseInt(args[0]);
  int y1 = Integer.parseInt(args[1]);
  int x2 = Integer.parseInt(args[2]);
  int y2 = Integer.parseInt(args[3]);
  Point p1 = new Point(x1, y1);
  Point p2 = new Point(x2, y2);
  double d = distance(p1, p2);
  if (d < 10) p1.color = RED;
  else p1.color = GREEN;
  System.out.println("Distance from "+p1+
                                 " to "+p2+" is "+d);
}

static double distance(Point a, Point b) {
  int dx = a.x - b.x;
  int dy = a.y - b.y;
  return Math.sqrt(dx*dx + dy*dy);
}
```



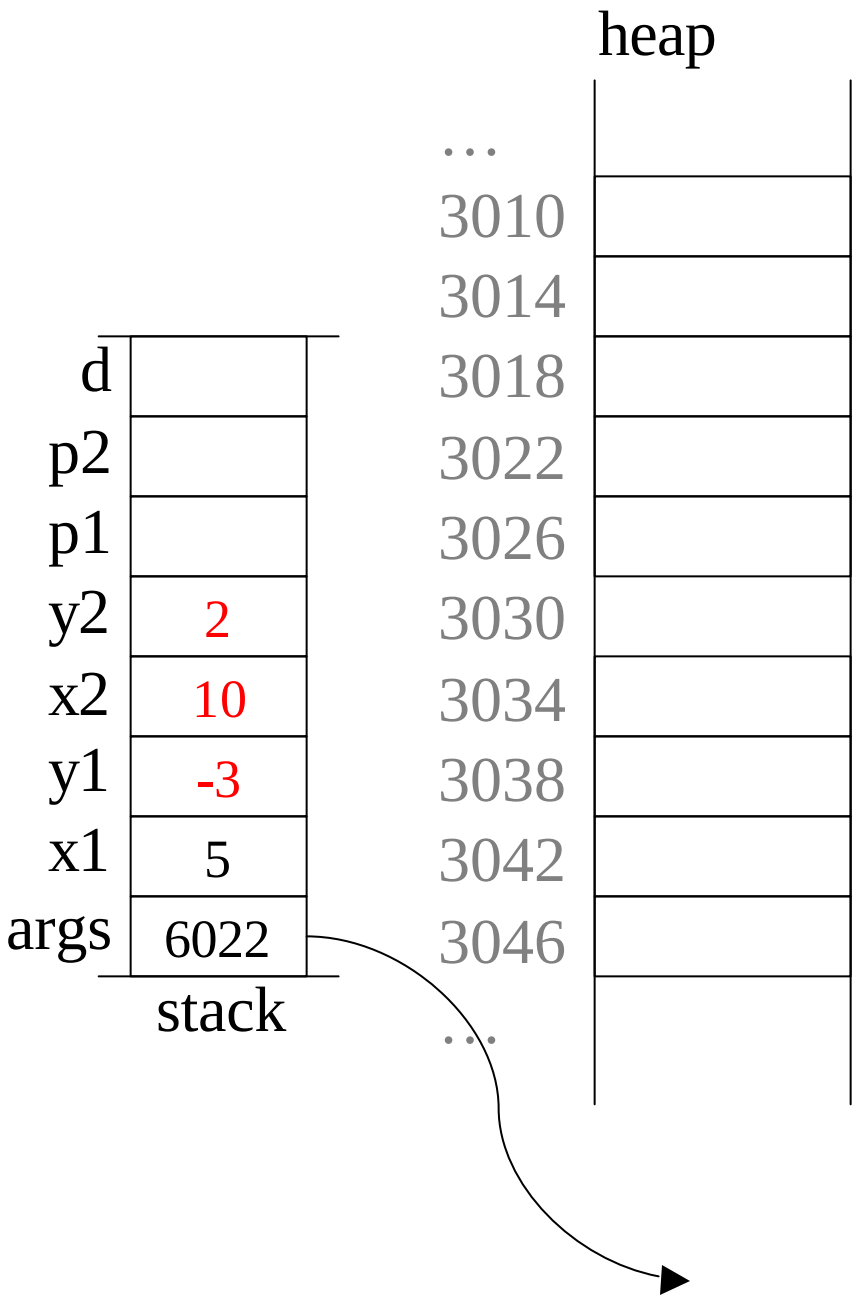
- Main is called
  - reference to an array of Strings is on top of stack
  - stack frame allocated for other variables

```
public static void main(String args[ ]) {
  int x1 = Integer.parseInt(args[0]);
  int y1 = Integer.parseInt(args[1]);
  int x2 = Integer.parseInt(args[2]);
  int y2 = Integer.parseInt(args[3]);
  Point p1 = new Point(x1, y1);
  Point p2 = new Point(x2, y2);
  double d = distance(p1, p2);
  if (d < 10) p1.color = RED;
  else p1.color = GREEN;
  System.out.println("Distance from "+p1+
                                 " to "+p2+" is "+d);
}

static double distance(Point a, Point b) {
  int dx = a.x - b.x;
  int dy = a.y - b.y;
  return Math.sqrt(dx*dx + dy*dy);
}
```



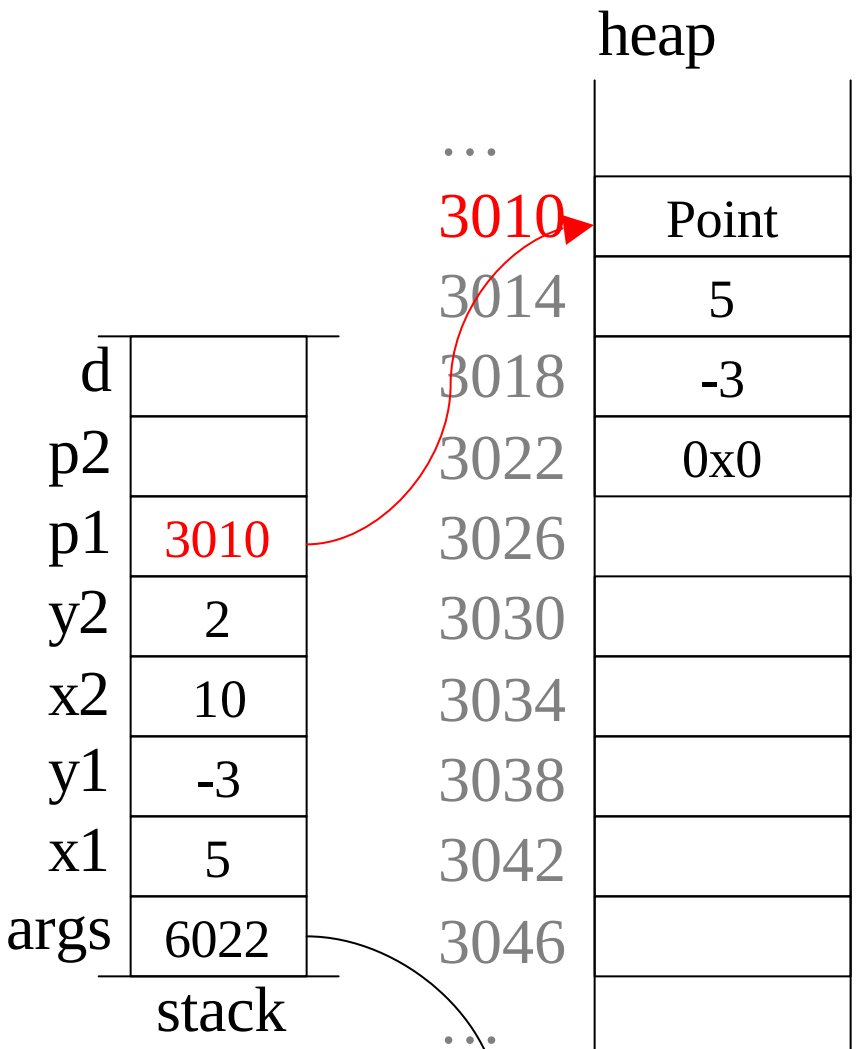
- Uses a *static method* in class Integer to convert String to integer

```
public static void main(String args[ ]) {
  int x1 = Integer.parseInt(args[0]);
  int y1 = Integer.parseInt(args[1]);
  int x2 = Integer.parseInt(args[2]);
  int y2 = Integer.parseInt(args[3]);
  Point p1 = new Point(x1, y1);
  Point p2 = new Point(x2, y2);
  double d = distance(p1, p2);
  if (d < 10) p1.color = RED;
  else p1.color = GREEN;
  System.out.println("Distance from "+p1+
                                  " to "+p2+" is "+d);
}

static double distance(Point a, Point b) {
  int dx = a.x - b.x;
  int dy = a.y - b.y;
  return Math.sqrt(dx*dx + dy*dy);
}
```



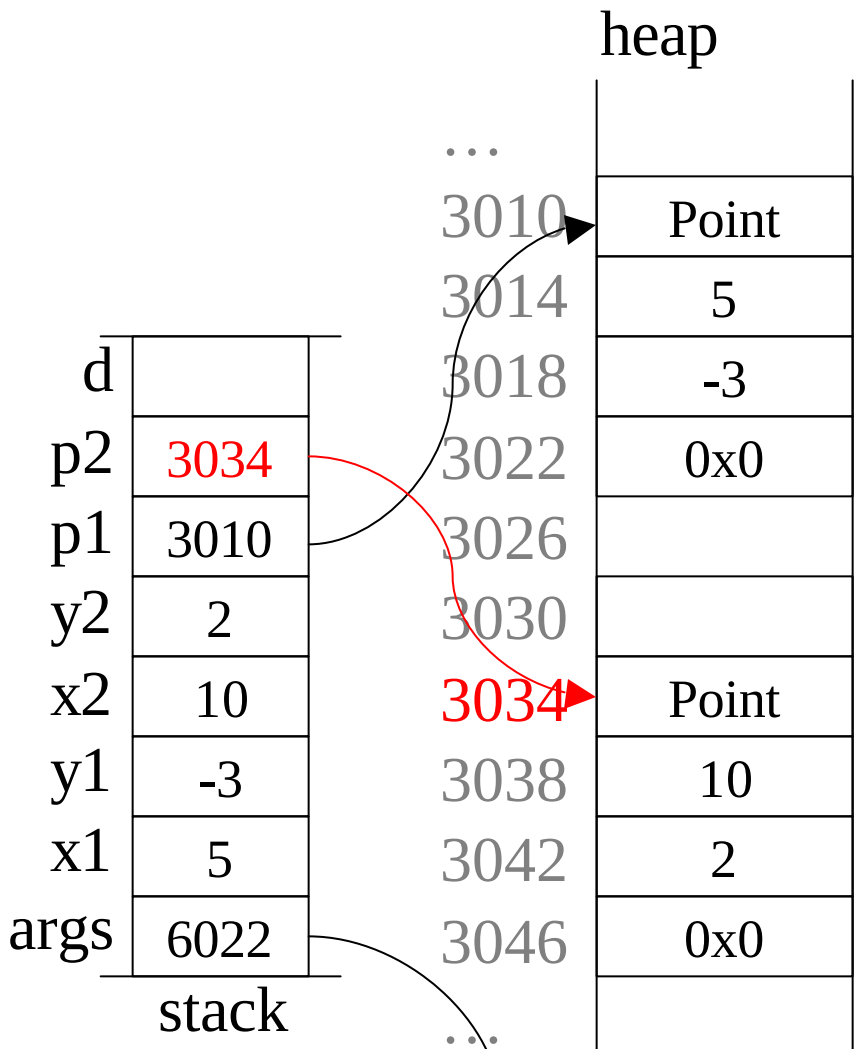
- Remaining args parsed

```
public static void main(String args[ ]) {
  int x1 = Integer.parseInt(args[0]);
  int y1 = Integer.parseInt(args[1]);
  int x2 = Integer.parseInt(args[2]);
  int y2 = Integer.parseInt(args[3]);
  Point p1 = new Point(x1, y1);
  Point p2 = new Point(x2, y2);
  double d = distance(p1, p2);
  if (d < 10) p1.color = RED;
  else p1.color = GREEN;
  System.out.println("Distance from "+p1+
                                  " to "+p2+" is "+d);
}

static double distance(Point a, Point b) {
  int dx = a.x - b.x;
  int dy = a.y - b.y;
  return Math.sqrt(dx*dx + dy*dy);
}
```



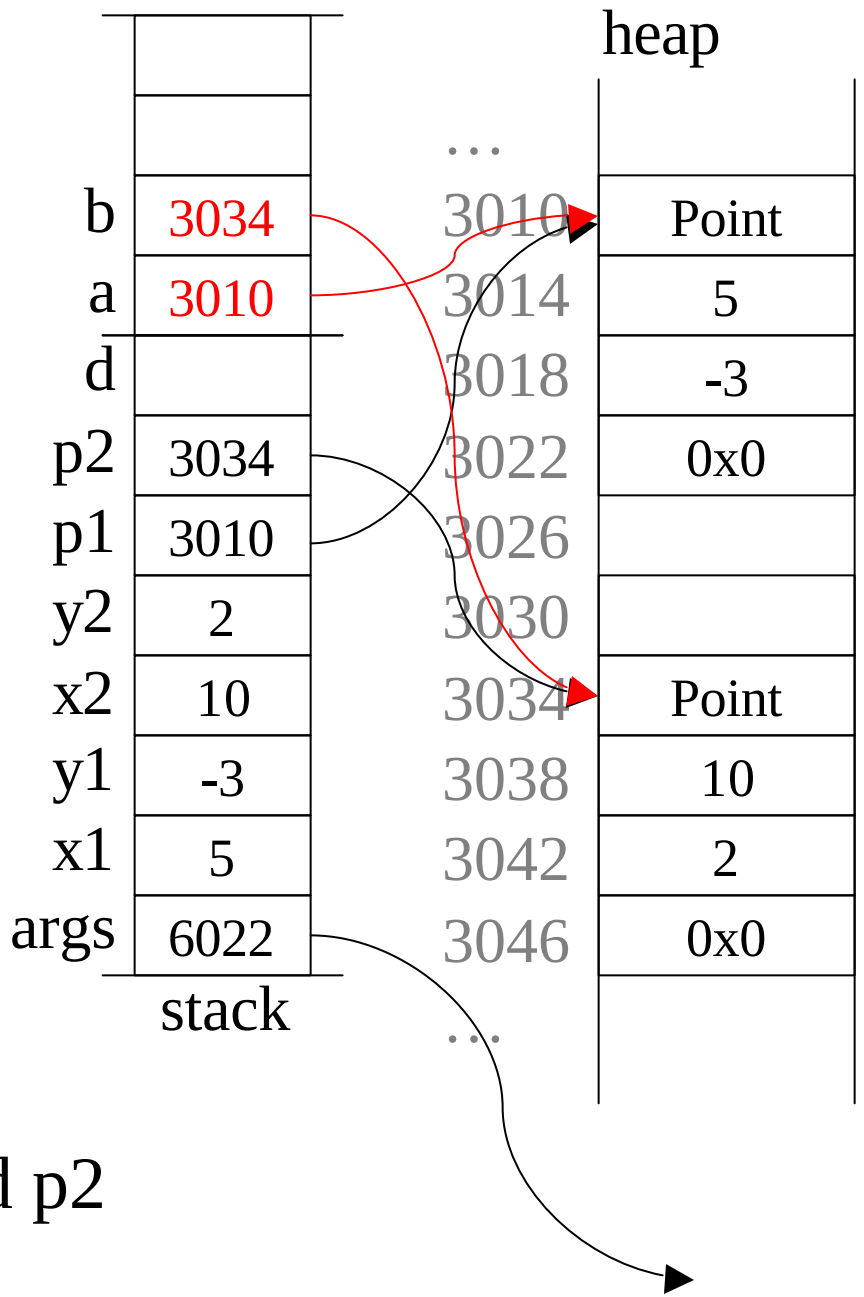
- Creates a new Point object on heap
- Initializes it to have coordinates x1, y1, default color
- Assigns p1 to refer to the new object

```
public static void main(String args[ ]) {
  int x1 = Integer.parseInt(args[0]);
  int y1 = Integer.parseInt(args[1]);
  int x2 = Integer.parseInt(args[2]);
  int y2 = Integer.parseInt(args[3]);
  Point p1 = new Point(x1, y1);
  Point p2 = new Point(x2, y2);
  double d = distance(p1, p2);
  if (d < 10) p1.color = RED;
  else p1.color = GREEN;
  System.out.println("Distance from "+p1+
                                  " to "+p2+" is "+d);
}

static double distance(Point a, Point b) {
  int dx = a.x - b.x;
  int dy = a.y - b.y;
  return Math.sqrt(dx*dx + dy*dy);
}
```



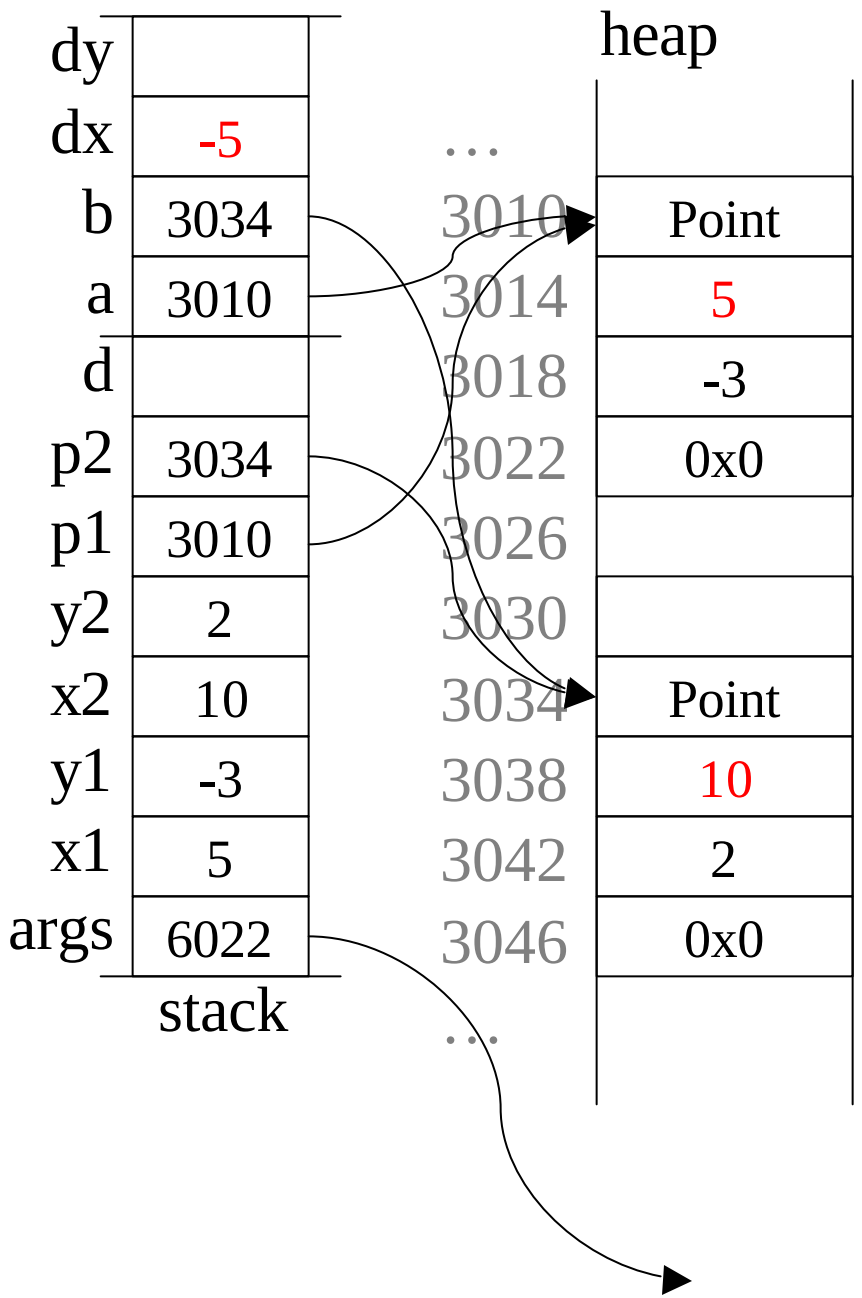
- Creates a second Point object on heap
- Initializes it to have coordinates x2, y2, default color
- Assigns p2 to refer to the new object

```
public static void main(String args[ ]) {
  int x1 = Integer.parseInt(args[0]);
  int y1 = Integer.parseInt(args[1]);
  int x2 = Integer.parseInt(args[2]);
  int y2 = Integer.parseInt(args[3]);
  Point p1 = new Point(x1, y1);
  Point p2 = new Point(x2, y2);
  double d = distance(p1, p2);
  if (d < 10) p1.color = RED;
  else p1.color = GREEN;
  System.out.println("Distance from "+p1+
                                  " to "+p2+" is "+d);
}

static double distance(Point a, Point b) {
  int dx = a.x - b.x;
  int dy = a.y - b.y;
  return Math.sqrt(dx*dx + dy*dy);
}
```



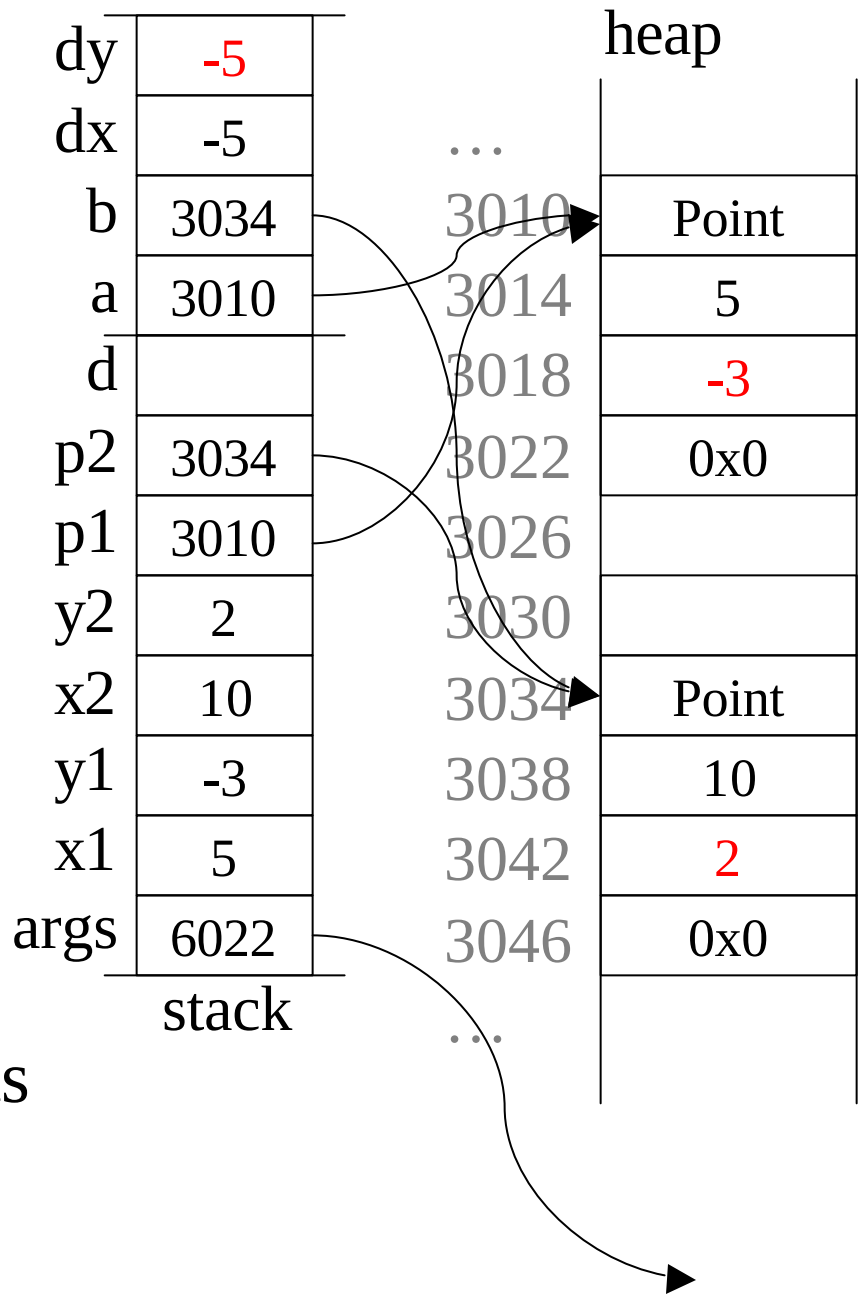
- calling distance( )
  - Pushes parameters p1 and p2
  - These become a and b

```
public static void main(String args[ ]) {
  int x1 = Integer.parseInt(args[0]);
  int y1 = Integer.parseInt(args[1]);
  int x2 = Integer.parseInt(args[2]);
  int y2 = Integer.parseInt(args[3]);
  Point p1 = new Point(x1, y1);
  Point p2 = new Point(x2, y2);
  double d = distance(p1, p2);
  if (d < 10) p1.color = RED;
  else p1.color = GREEN;
  System.out.println("Distance from "+p1+
                                 " to "+p2+" is "+d);
}

static double distance(Point a, Point b) {
  int dx = a.x - b.x;
  int dy = a.y - b.y;
  return Math.sqrt(dx*dx + dy*dy);
}
```



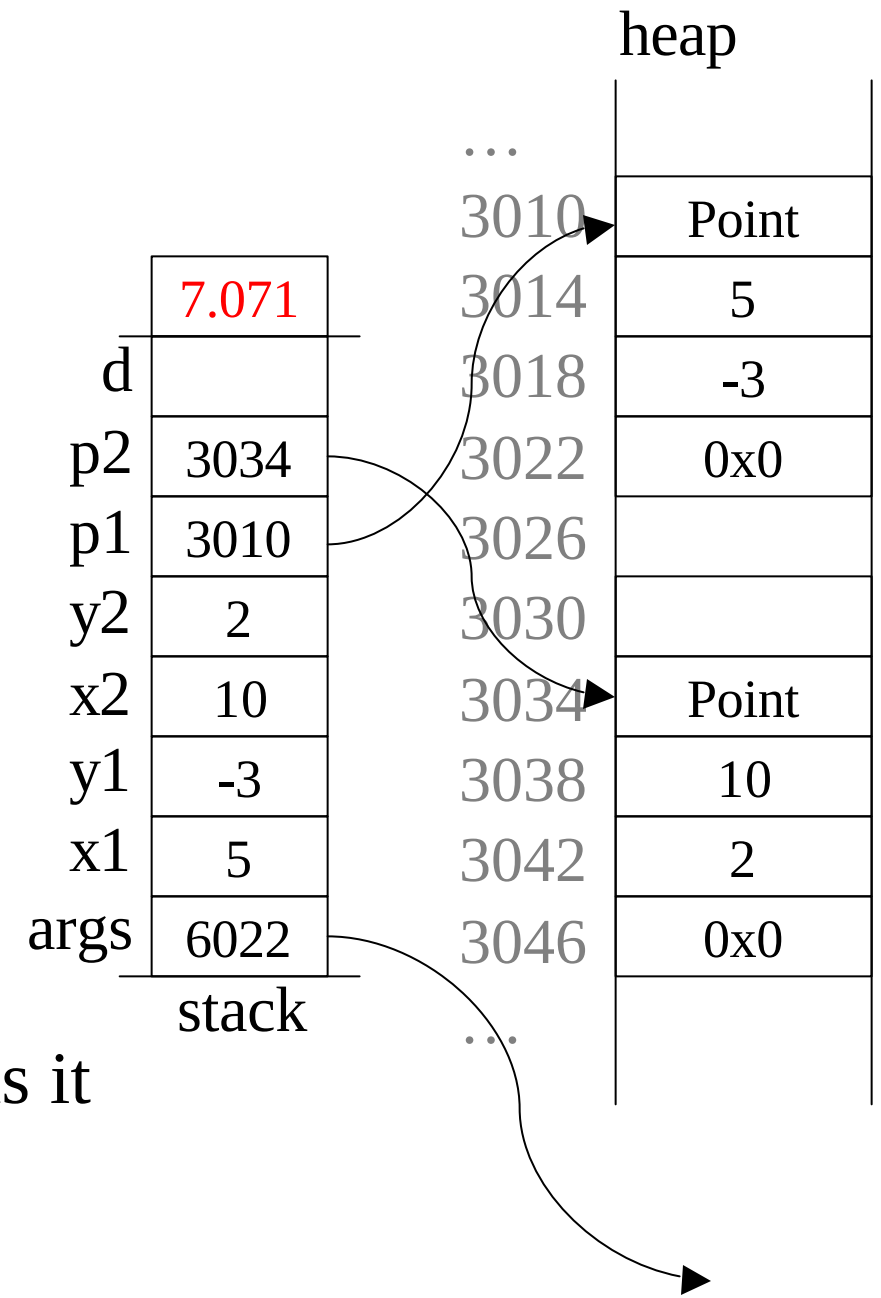
- Accesses objects pointed to by a and b
  - reads the *field x* from each

```
public static void main(String args[ ]) {
  int x1 = Integer.parseInt(args[0]);
  int y1 = Integer.parseInt(args[1]);
  int x2 = Integer.parseInt(args[2]);
  int y2 = Integer.parseInt(args[3]);
  Point p1 = new Point(x1, y1);
  Point p2 = new Point(x2, y2);
  double d = distance(p1, p2);
  if (d < 10) p1.color = RED;
  else p1.color = GREEN;
  System.out.println("Distance from "+p1+
                                 " to "+p2+" is "+d);
}

static double distance(Point a, Point b) {
  int dx = a.x - b.x;
  int dy = a.y - b.y;
  return Math.sqrt(dx*dx + dy*dy);
}
```



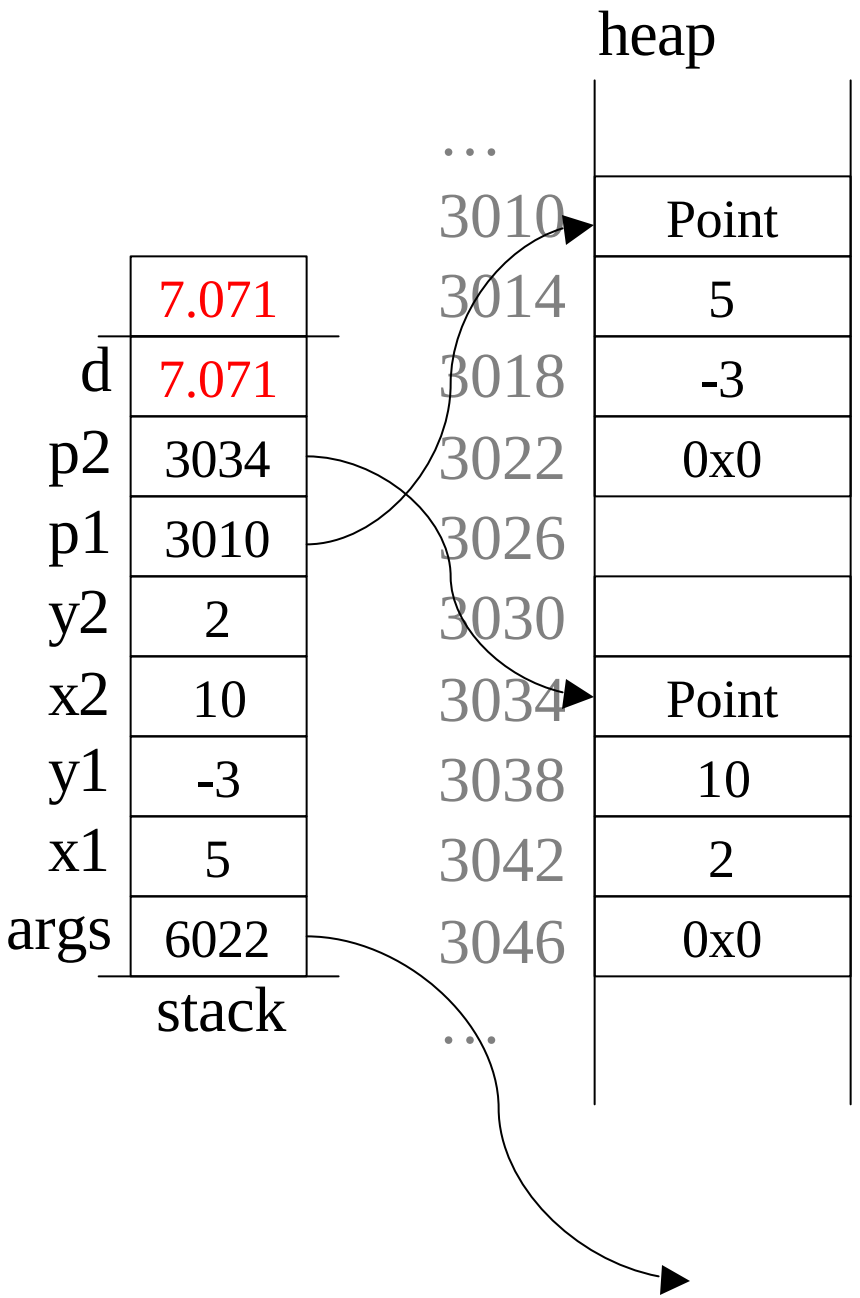
- Accesses *field y* from objects

```
public static void main(String args[ ]) {
  int x1 = Integer.parseInt(args[0]);
  int y1 = Integer.parseInt(args[1]);
  int x2 = Integer.parseInt(args[2]);
  int y2 = Integer.parseInt(args[3]);
  Point p1 = new Point(x1, y1);
  Point p2 = new Point(x2, y2);
  double d = distance(p1, p2);
  if (d < 10) p1.color = RED;
  else p1.color = GREEN;
  System.out.println("Distance from "+p1+
                                  " to "+p2+" is "+d);
}

static double distance(Point a, Point b) {
  int dx = a.x - b.x;
  int dy = a.y - b.y;
  return Math.sqrt(dx*dx + dy*dy);
}
```



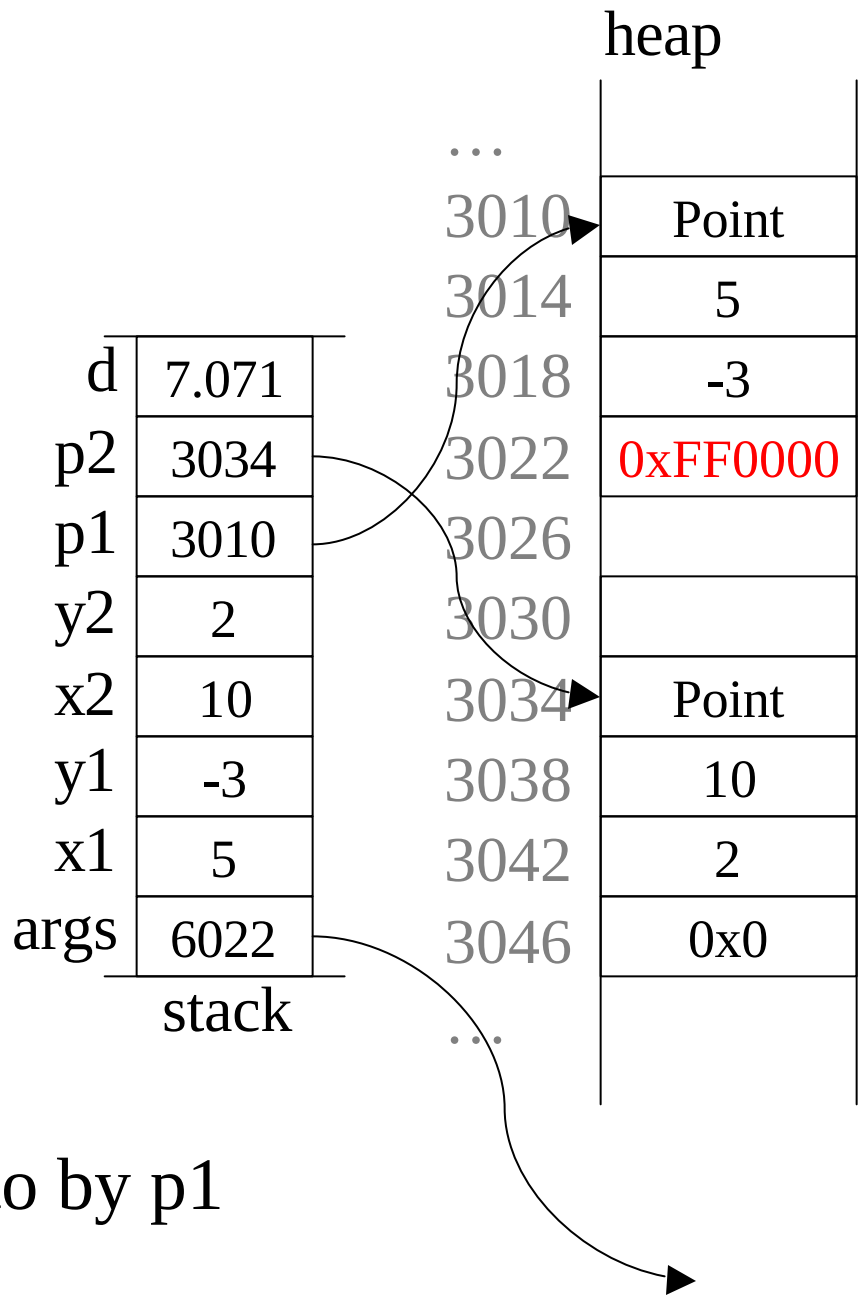
- Computes result, and returns it

```
public static void main(String args[ ]) {
  int x1 = Integer.parseInt(args[0]);
  int y1 = Integer.parseInt(args[1]);
  int x2 = Integer.parseInt(args[2]);
  int y2 = Integer.parseInt(args[3]);
  Point p1 = new Point(x1, y1);
  Point p2 = new Point(x2, y2);
  double d = distance(p1, p2);
  if (d < 10) p1.color = RED;
  else p1.color = GREEN;
  System.out.println("Distance from "+p1+
                                  " to "+p2+" is "+d);
}

static double distance(Point a, Point b) {
  int dx = a.x - b.x;
  int dy = a.y - b.y;
  return Math.sqrt(dx*dx + dy*dy);
}
```



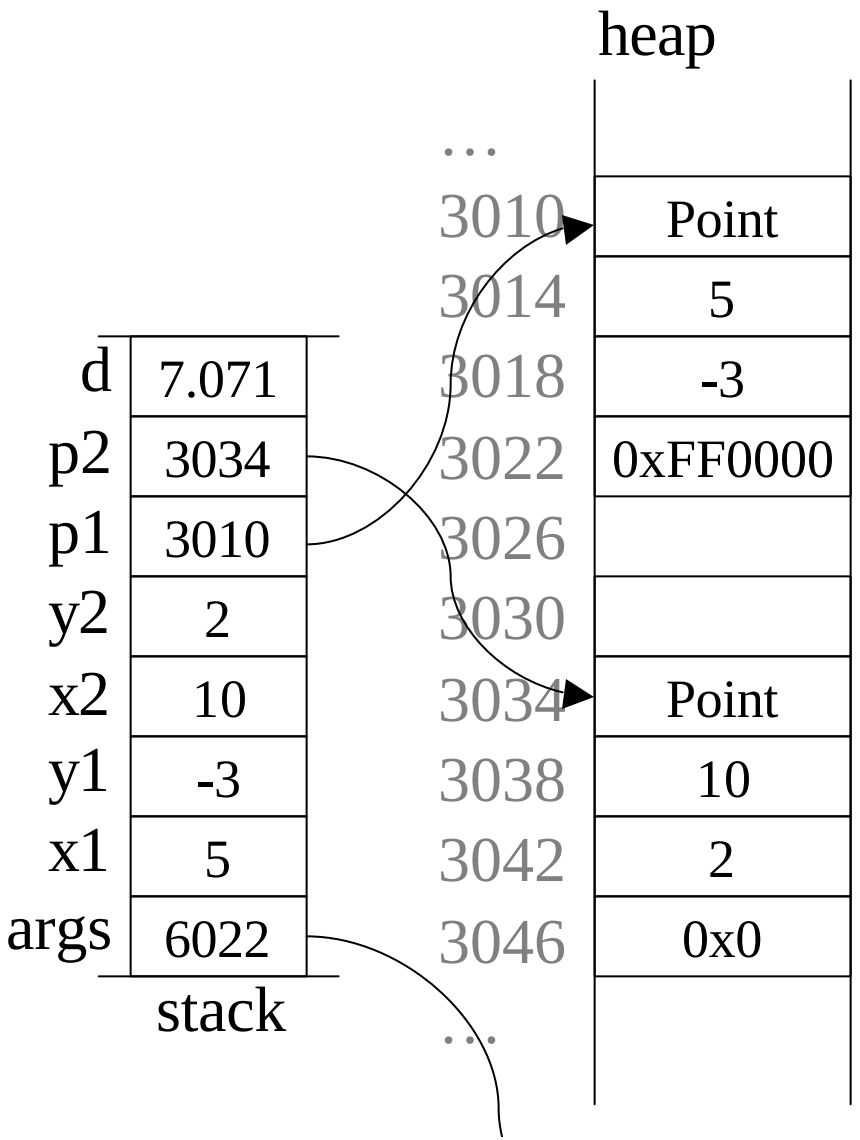
- Assigns return value to d

```
public static void main(String args[ ]) {
  int x1 = Integer.parseInt(args[0]);
  int y1 = Integer.parseInt(args[1]);
  int x2 = Integer.parseInt(args[2]);
  int y2 = Integer.parseInt(args[3]);
  Point p1 = new Point(x1, y1);
  Point p2 = new Point(x2, y2);
  double d = distance(p1, p2);
  if (d < 10) p1.color = RED;
  else p1.color = GREEN;
  System.out.println("Distance from "+p1+
                                  " to "+p2+" is "+d);
}

static double distance(Point a, Point b) {
  int dx = a.x - b.x;
  int dy = a.y - b.y;
  return Math.sqrt(dx*dx + dy*dy);
}
```



- Compares d to 10
- Then access object pointed to by p1 and sets color to RED

```
public static void main(String args[ ]) {
  int x1 = Integer.parseInt(args[0]);
  int y1 = Integer.parseInt(args[1]);
  int x2 = Integer.parseInt(args[2]);
  int y2 = Integer.parseInt(args[3]);
  Point p1 = new Point(x1, y1);
  Point p2 = new Point(x2, y2);
  double d = distance(p1, p2);
  if (d < 10) p1.color = RED;
  else p1.color = GREEN;
  System.out.println("Distance from "+p1+
                                  " to "+p2+" is "+d);
}

static double distance(Point a, Point b) {
  int dx = a.x - b.x;
  int dy = a.y - b.y;
  return Math.sqrt(dx*dx + dy*dy);
}
```



- Prints results

```
Welcome to DrJava.
> java Distance 5 -3 10 2
Distance from (5, -3, 0xff0000) to (10, 2, 0x0) is 7.071067811865475
```

Lessons

- In Java, almost everything is an object:
  - e.g., String, array of whatever, File, Point
  - but not primitive types (int, boolean, char, …)
- Objects have contents:
  - e.g., Point has x, y, and color
- Only way to access object contents is through a reference
  - Can have multiple references to same object
  - No direct way to figure out the actual address from within a Java program; no way to turn an address (if you had one) back in to a reference.

References as Parameters

- Method parameters are a copy of the *reference*, **not the object**
  - Not always intuitive! Consider:

```
static void swap(Point p1, Point p2) {
  Point tmp = p2;
  p2 = p1;
  p1 = tmp;
}
```

```
static void destroy(Point p1) {
  p1.color = 0x0;
  p1 = null;
}
```

  - (But the alternative is even more confusing)

Assignment

- Same as parameter passing (or rather, vice-versa)
- Assigns copy of the *reference*, **does not make a copy of the object**
  - Not always intuitive! Consider:

```
Button noButton = new Button("No");
Button yesButton = new Button();
yesButton = noButton;
yesButton.setLabel("Yes");
panel.add(noButton);
panel.add(yesButton);
```

  - Called *aliasing*: often confusing!
  - There *is* a way to make a *copy* of an object; rarely used

Comparing References

- Compares the *references* for equality, **not the objects**
  - Not always intuitive! Consider:

```
Point p1 = new Point(5, 3);
Point p2 = new Point(5, 3);
if (p1 == p2)  System.out.println("same reference");
```

  - Same goes for '!='
  - If you want to compare *objects* for equality, instead do:

```
if (p1.equals(p2))  System.out.println("identical objects");
```

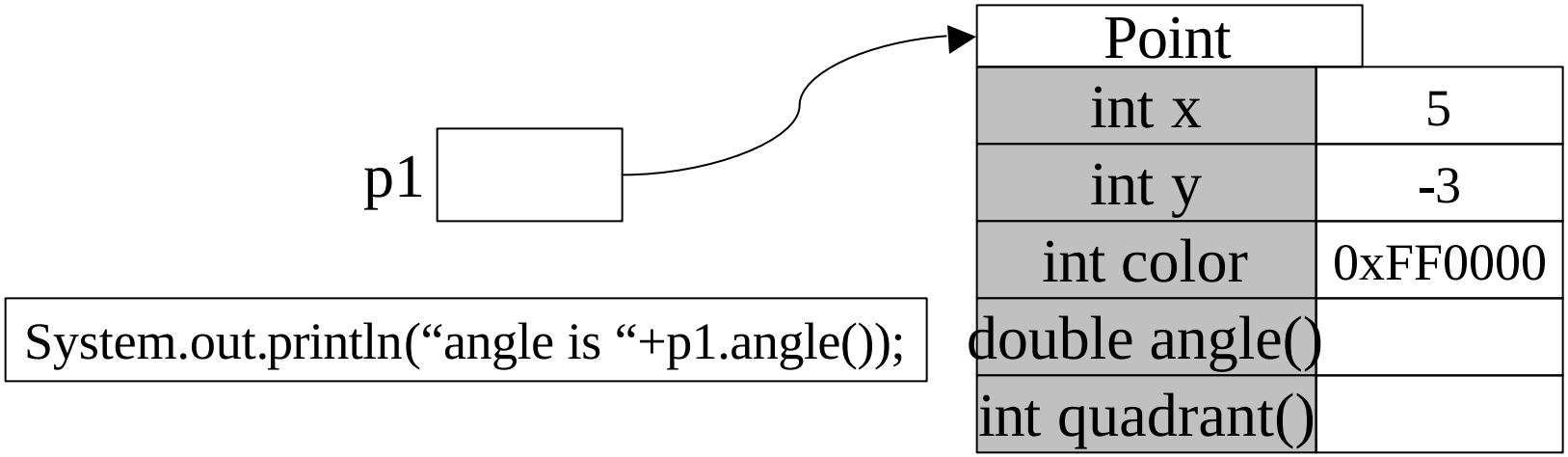
Live and Dead Objects

```
Button noButton = new Button("No");
Button yesButton = new Button();
yesButton = noButton;
```

- Question: What happens to second Button instance?
  - Lost forever; call it *dead*
- Live objects (a recursive definition)
  - Anything referred to by another live object
- Dead objects
  - the rest
  - Eventually reclaimed, or *garbage collected*, by the system

Method Dispatching

- Objects have *instance methods* as well as fields



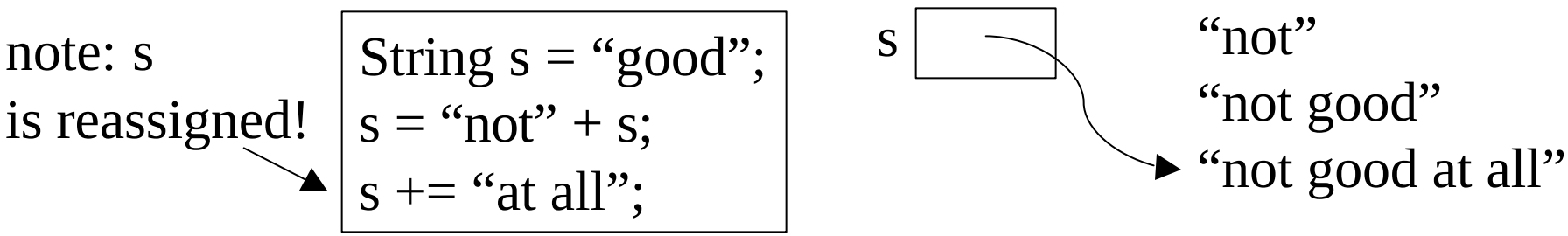
- Method dispatching: process of finding the right method to call
  - Dynamic: looks at object, searches for method with correct name and parameter types.

String

- Like any object, but:
  - has a special way of creating new ones: "…"
    - "abc" creates a new String, returns reference to it
    - has a special operator for concatenation: +
- Immutable
  - All of it's *fields* are declared *private*, so are inaccessible
  - None of its methods ever change the object
  - No one can create additional methods for String
  - Thought exercise: why is this important?

String Concatenation

- If either lhs or rhs is a (reference to a) String, then '+' means string concatenation; also use "+="
- String concatenation creates a new string object and returns a reference to it



- If one side not a String, it is converted:
  - e.g., "abc" + 5 → "abc5"
    1 + (2 + "3") → "123"

String Comparison

- Operators “==“ and “!=“ compare references
  - Rarely useful for strings
  - Usually want to compare contents
  - So use equals() instead

```
String s1 = “Hello”;
if (s1 == “Hello”) System.out.println(“oops”);
if (s1.equals(“Hello”) System.out.println(“better”);
if (“Hello”.equals(s1)) System.out.println(“odd, but okay”);
```

Other String Methods

- Other than equals(), there are many
  - Look in Java API (on web)
- Inspecting a string:
  - “Hello”.length() → returns int *5*
  - “Hello”.charAt(1) → returns char ‘e’
  - “Hello”.indexOf(‘o’) → returns int *4*
  - “Hello”.endsWith(“lo”) → returns boolean *true*
- Modifying a string (hey, aren’t they immutable!):
  - “cs211”.toUpperCase() → returns String “CS211”
  - “cs211”.replace(‘1’, ‘0’) → returns String “cs200”
  - “ cs211 “.trim() → returns String “cs211”

Arrays

- Read Weiss sec. 2.4
- Arrays are objects

```
int [ ] a = new int[5];
a[0] = 123;
a[4] = a.length;
```

- Arrays can hold references as well as primitives

```
String [ ] staff = new String[5]; // initially, all null
staff[0] = “Kevin”; // new object, save reference in array
String [ ] people = staff; // alias to same array!
people[1] = “Eli”; // new object, save reference in array
```