Why do we have to write a type for every variable in Java?

Why do we have to write a type for every variable in Java?

```
def first_letter(s):
    if len(s) > 0:
        return s[0]
    else:
        return ""


# your test cases – everything is ok!
print first_letter("cs2230")
print first_letter("hello world")
print first_letter("")

# the user's input
print first_letter(4)

Traceback (most recent call last):  File "types.py", line 16, in
<module>    print first_letter(4)   # breaks!  File "types.py",
line 4, in first_letter    if len(s) > 0:
TypeError: object of type 'int' has no len()
```

```java
public class Types {
    public static void main(String[] args) {
        first_letter("cs2230");
        first_letter("hello world");
        first_letter("");
```

giving it an integer

```java
        first_letter(4);
    }
```

wants a String

```java
    public static String first_letter(String s) {
        if (s.length() > 0) {
            return s.substring(0,1);
        } else {
            return "";
        }
    }
}
```

Before we run the Java program, we compile it. And the compiler says:

```
Types.java:7: error: incompatible types: int cannot be converted to String
first_letter(4);
```

socrative.com
room: CS2230

Peer instruction

```
public class Mystery {
    public static void main(String[] args) {
        int f = 22;
        int g = 1000;
        f = g + f;
    }


    public static int stuff(int x) {
        return -x
    }
}
```

What is the result of trying to compile and run this program?

a) runs fine
b) error while compiling (“compile time error”)
c) error while running (“runtime error”)

Today’s big ideas

- Two kinds of data in Java: **primitives** and **objects**
- We refer to an object using a **reference**
- There is a difference between passing objects and primitives to a method

CS 2230
CS II: Data structures

Meeting 3: Objects and references

Brandon Myers

University of Iowa

An example from the doctor

- We need to build a system to track patients
- First, patients have a name and height (in inches)

```
class Patient {
    String name;
    int height;
}
```

defines the **class** Patient

Creating a patient

```java
class Patient {
    String name;
    int height;

    // constructor (says how to initialize a new Patient)
    Patient(String n, int h) {
        name = n;
        height = h;
    }

    public static void main(String[] args) {
        // create a Patient
       Patient p1 = new Patient("Jane Doe", 65);
    }
}
```

Updating a patient

When patients come it for a check up, we want to update their height with the latest measurement

```
class Patient {
    String name;
    int height;

    Patient(String n, int h) {
        name = n;
        height = h;
    }

    void updateHeight(int newHeight) {
        height = newHeight;
    }

    public static void main(String[] args) {
        Patient p1 = new Patient("Jane Doe", 65);
        p1.updateHeight(70);
    }
}
```

method

call the method on p1

https://b.socrative.com/login/student/

Peer instruction

```
public class MathStuff {
    static void squareIt(int x) {
        x = x*x;
    }

    public static void main(String[] args) {
        int a = 10;
        squareIt(a);
        System.out.println(a);
    }
}
```

What does the program print to the console?

a) a
b) 10
c) 100
d)
e) 10*10

https://b.socrative.com/login/student/

Peer instruction

```
public class MathStuff {
    static void squareIt(int[] x) {
        x[0] = x[0] * x[0];
    }

    public static void main(String[] args) {
        int[] a = new int[3];
        a[0] = 10;
        squareIt(a);
        System.out.println(a[0]);
    }
}
```

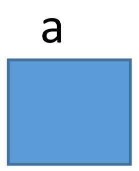
What does the program print to the console?

a) a
b) 10
c) 100
d)
e) x[0]*x[0]

let's work through that last example with diagrams

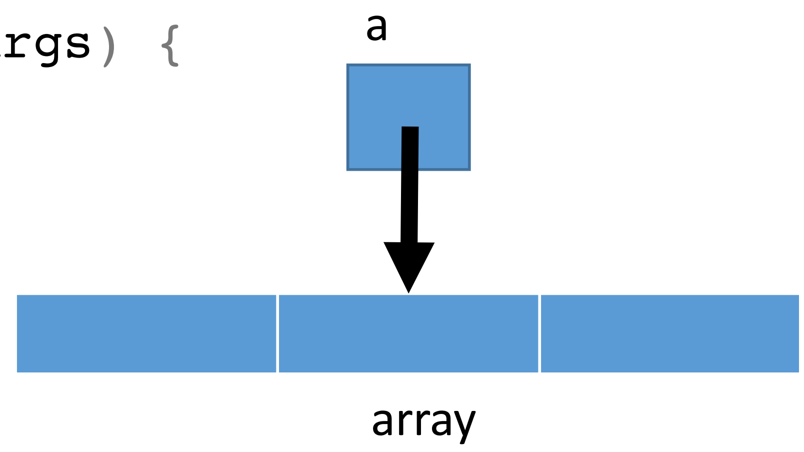
References

```
public class MathStuff {
    static void squareIt(int[] x) {
        x[0] = x[0] * x[0];
    }

    public static void main(String[] args) {
      > int[] a;
        a = new int[3];
        a[0] = 10;
        squareIt(a);
        System.out.println(a[0]);
    }
}
```



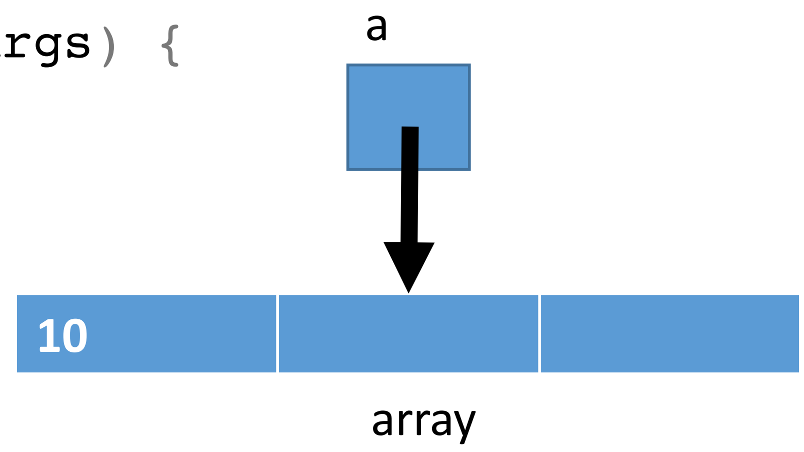
References

```
public class MathStuff {
    static void squareIt(int[] x) {
        x[0] = x[0] * x[0];
    }

    public static void main(String[] args) {
        int[] a;
      > a = new int[3];
        a[0] = 10;
        squareIt(a);
        System.out.println(a[0]);
    }
}
```



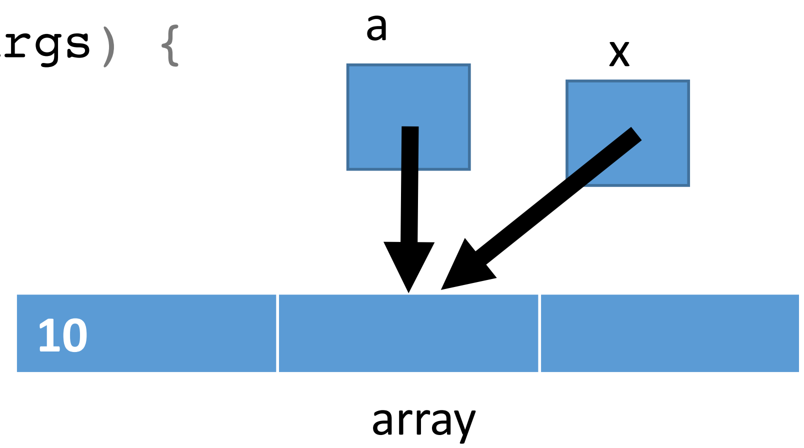
References

```
public class MathStuff {
    static void squareIt(int[] x) {
        x[0] = x[0] * x[0];
    }

    public static void main(String[] args) {
        int[] a;
        a = new int[3];
      > a[0] = 10;
        squareIt(a);
        System.out.println(a[0]);
    }
}
```



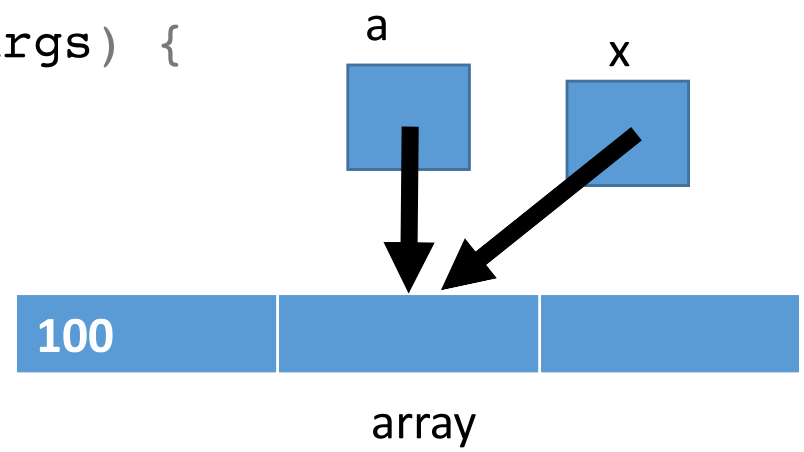
References

```
public class MathStuff {
    static void squareIt(int[] x) {
        x[0] = x[0] * x[0];
    }

    public static void main(String[] args) {
        int[] a;
        a = new int[3];
        a[0] = 10;
      > squareIt(a);
        System.out.println(a[0]);
    }
}
```



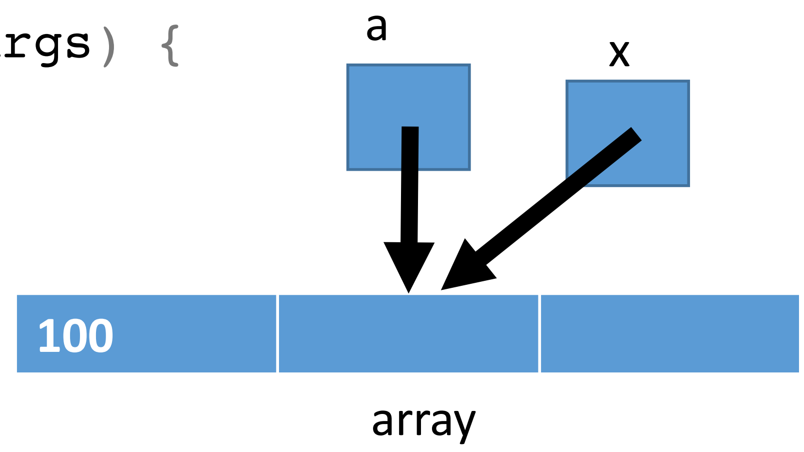
References

```
public class MathStuff {
    static void squareIt(int[] x) {
      >  x[0] = x[0] * x[0];
    }

    public static void main(String[] args) {
        int[] a;
        a = new int[3];
        a[0] = 10;
        squareIt(a);
        System.out.println(a[0]);
    }
}
```



References

```
public class MathStuff {
    static void squareIt(int[] x) {
        x[0] = x[0] * x[0];
    }

    public static void main(String[] args) {
        int[] a;
        a = new int[3];
        a[0] = 10;
        squareIt(a);
      > System.out.println(a[0]);
    }
}
```



Section today

Today’s big ideas

- Two kinds of data in Java: **primitives** and **objects**
- We refer to an object using a **reference**
- There is a difference between passing objects and primitives to a method

Today’s big ideas

- When we want an array of objects, we store their *references* in the array
- It is important to distinguish between the specification and implementation of a class
- **public** and **private** control access to fields and methods

CS 2230
CS II: Data structures

Meeting 3: Objects oriented programming

Brandon Myers

University of Iowa

Peer instruction

https://b.socrative.com/login/student/
CS2230A

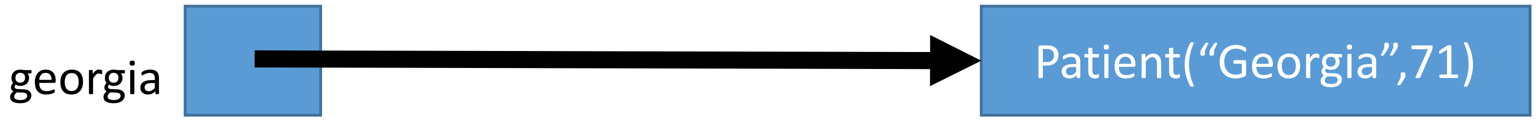
```
class Doctor {
    void checkup(Patient p) {
      Patient p2 = new Patient(p.name, p.height+10);
      p = p2;
    }


    public static void main(String[] args) {
      Doctor d = new Doctor();
      Patient georgia = new Patient("Georgia", 71);

      d.checkup(georgia);
      System.out.println(georgia.height);
    }
}
```

What does the program print to the console?

a) 81
b) georgia.height
c) Patient
d) Georgia
e) 71

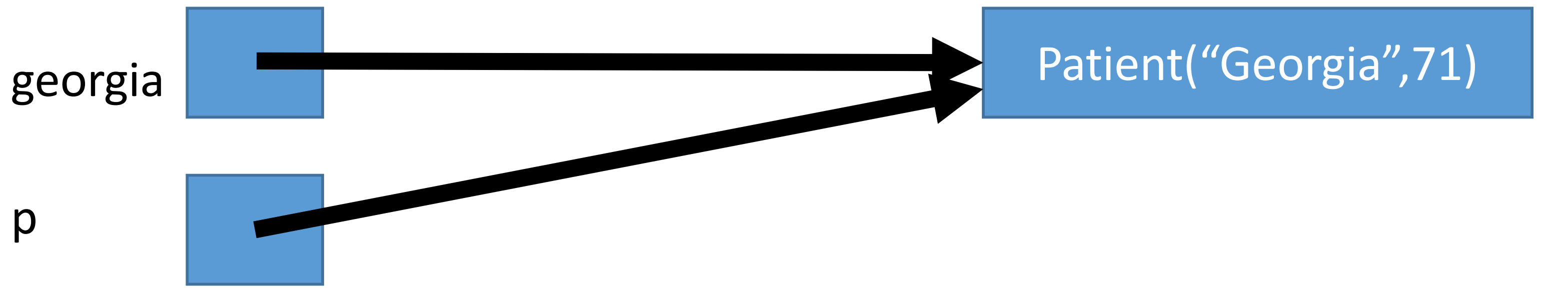
Peer instruction: explanation

```
class Doctor {
    void checkup(Patient p) {
      Patient p2 = new Patient(p.name, p.height+10);
      p = p2;
    }


    public static void main(String[] args) {
      Doctor d = new Doctor();
    > Patient georgia = new Patient("Georgia", 71);

      d.checkup(georgia);
      System.out.println(georgia.height);
    }
}
```



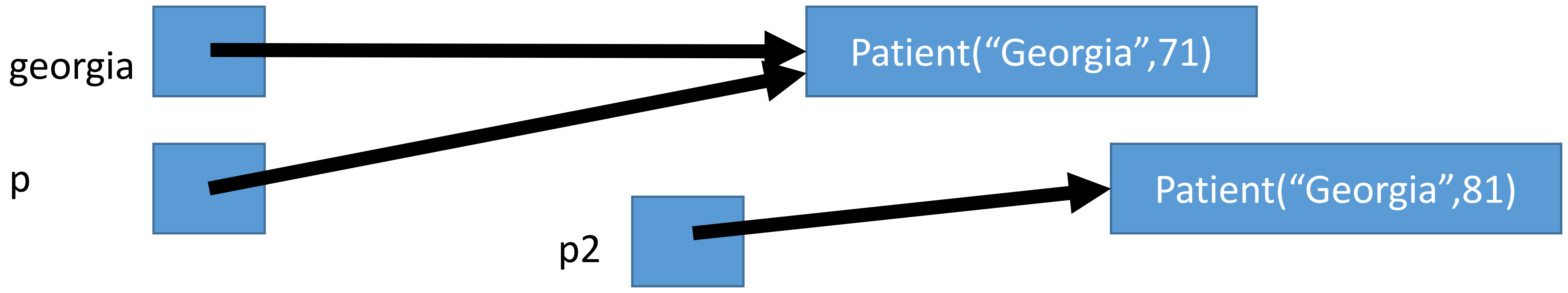
Peer instruction: explanation

```
class Doctor {
    void checkup(Patient p) {
      Patient p2 = new Patient(p.name, p.height+10);
      p = p2;
    }


    public static void main(String[] args) {
      Doctor d = new Doctor();
      Patient georgia = new Patient("Georgia", 71);

    > d.checkup(georgia);
      System.out.println(georgia.height);
    }
}
```



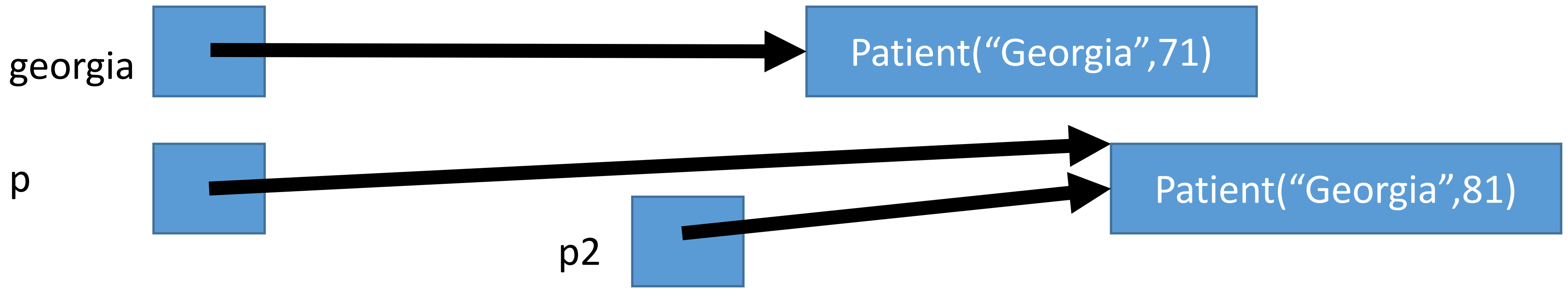
Peer instruction: explanation

```
class Doctor {
    void checkup(Patient p) {
    > Patient p2 = new Patient(p.name, p.height+10);
      p = p2;
    }


    public static void main(String[] args) {
      Doctor d = new Doctor();
      Patient georgia = new Patient("Georgia", 71);

      d.checkup(georgia);
      System.out.println(georgia.height);
    }
}
```



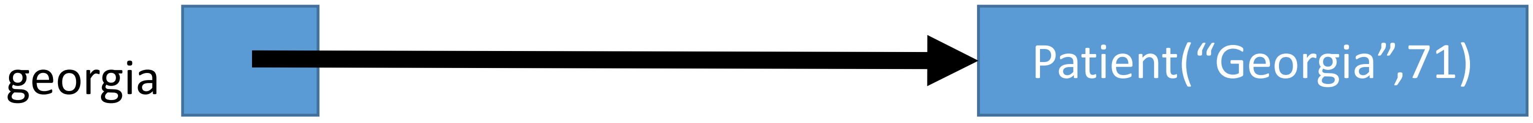
Peer instruction: explanation

```
class Doctor {
    void checkup(Patient p) {
      Patient p2 = new Patient(p.name, p.height+10);
    > p = p2;
    }


    public static void main(String[] args) {
      Doctor d = new Doctor();
      Patient georgia = new Patient("Georgia", 71);

      d.checkup(georgia);
      System.out.println(georgia.height);
    }
}
```



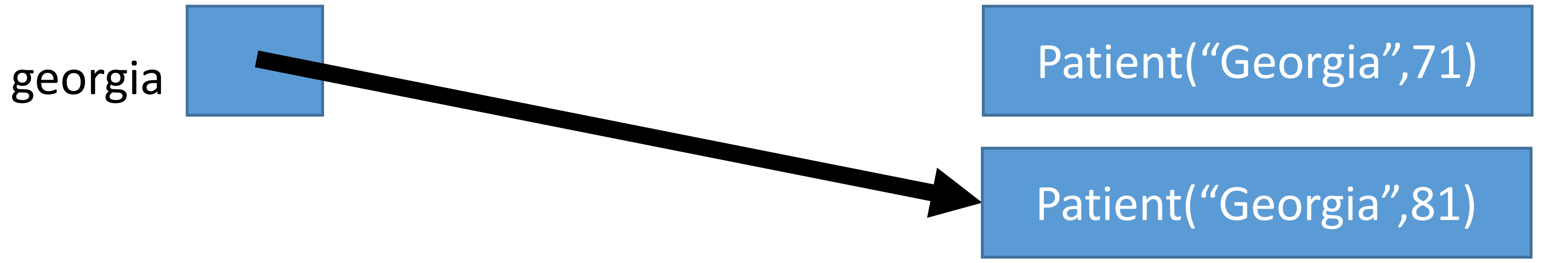
Peer instruction: explanation

```
class Doctor {
    void checkup(Patient p) {
      Patient p2 = new Patient(p.name, p.height+10);
      p = p2;
    }


    public static void main(String[] args) {
      Doctor d = new Doctor();
      Patient georgia = new Patient("Georgia", 71);

      d.checkup(georgia);
    > System.out.println(georgia.height);
    }
}
```



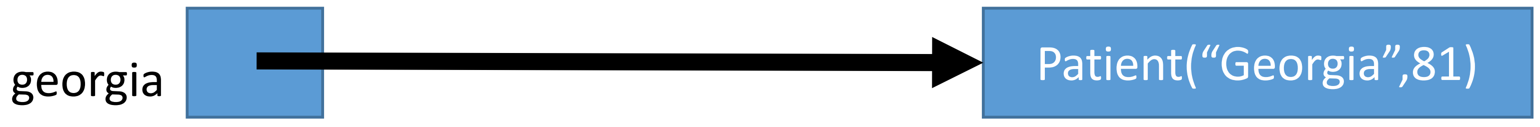
How do we fix the program? Option A

```java
class Doctor {
    Patient checkup(Patient p) {
      Patient p2 = new Patient(p.name, p.height+10);
      return p2;
    }


    public static void main(String[] args) {
      Doctor d = new Doctor();
      Patient georgia = new Patient("Georgia", 71);

      georgia = d.checkup(georgia);
      System.out.println(georgia.height);
    }
}
```



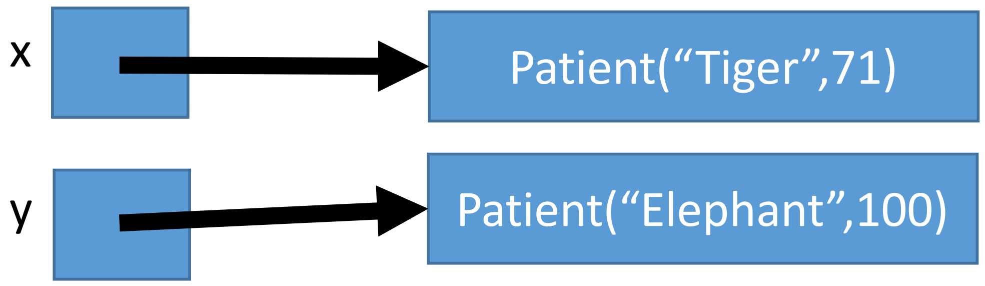
How do we fix the program? Option B

```
class Doctor {
    void checkup(Patient p) {
      p.height = p.height + 10;
    }


    public static void main(String[] args) {
      Doctor d = new Doctor();
      Patient georgia = new Patient("Georgia", 71);

      d.checkup(georgia);
      System.out.println(georgia.height);
    }
}
```

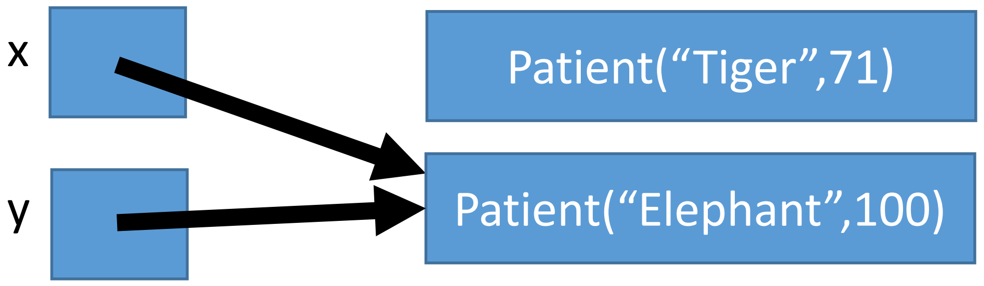


Trust the boxes and arrows!

Patient x = new Patient(“Tiger”,71)
Patient y = new Patient(“Elephant”,100)

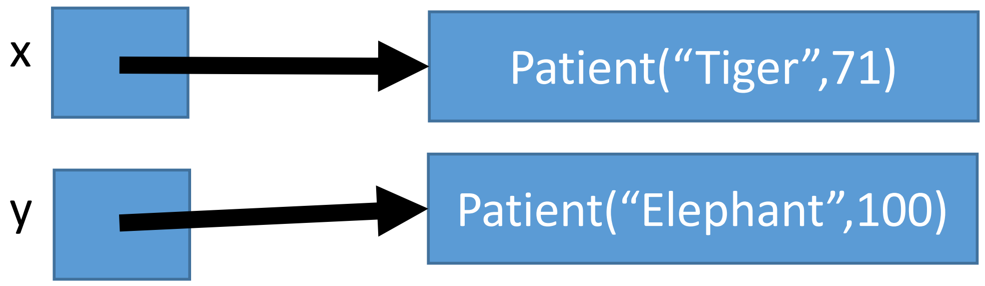


assigning to a reference
x = y



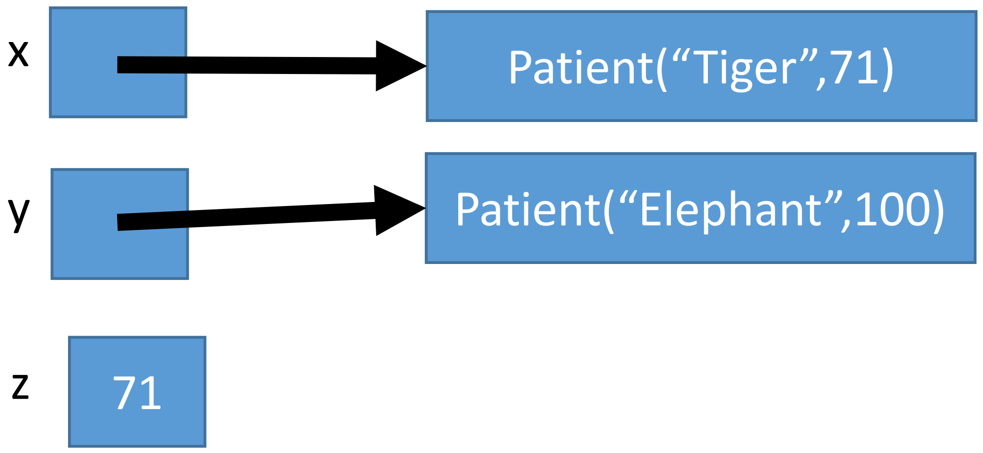
Trust the boxes and arrows!

```
Patient x = new Patient("Tiger",71)
Patient y = new Patient("Elephant",100)
```



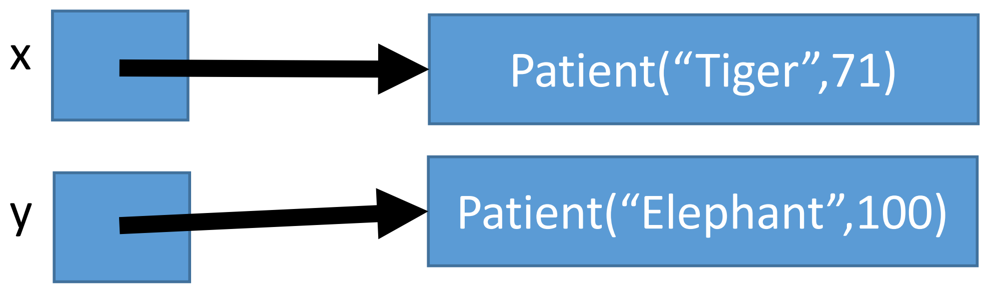
reading a field

```
int z = x.height;
```

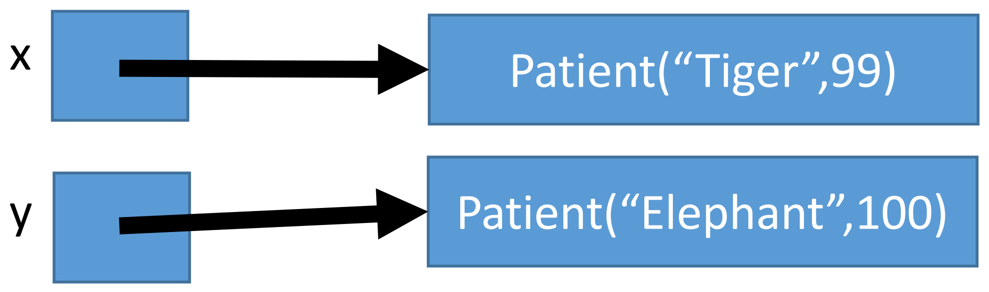


Trust the boxes and arrows!

Patient x = new Patient("Tiger",71)
Patient y = new Patient("Elephant",100)

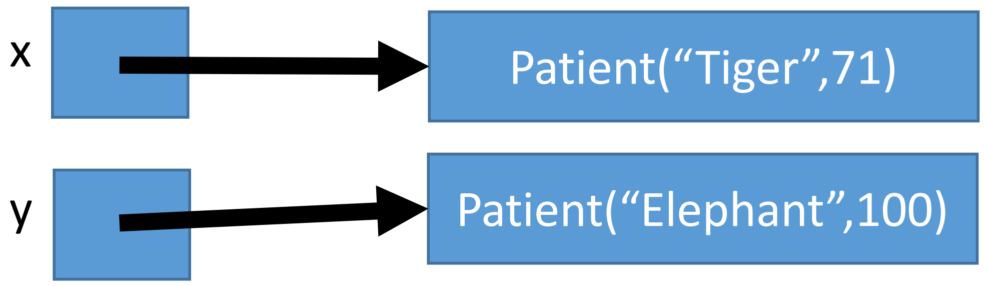


writing a field
x.height = 99;

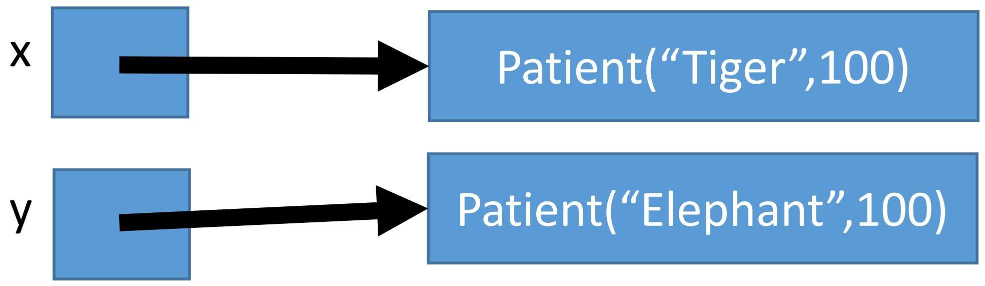


Trust the boxes and arrows!

Patient x = new Patient("Tiger",71)
Patient y = new Patient("Elephant",100)



reading a field and writing another field
x.height = y.height;



Extend the patients application

- Now we want a database of Patients that can do two things
  - Register a new Patient in the database (if we have room)
  - Print our Patients’ names in alphabetical order

Specification of the PatientDatabase class

```
class PatientDatabase {
// Register a new Patient in the database
// return false if out of space
  boolean registerNewPatient(String name) { ... }

// Print all patient names in alphabetical order
  void printNamesAlphabetically() { ... }

  public static void main(String[] args) {
    PatientDatabase db = new PatientDatabase(100);
    db.registerNewPatient("Ron");
    db.registerNewPatient("Hermoine");
    db.registerNewPatient("Snape");
    db.registerNewPatient("Harry");
    db.printNamesAlphabetically();
 }
}
```

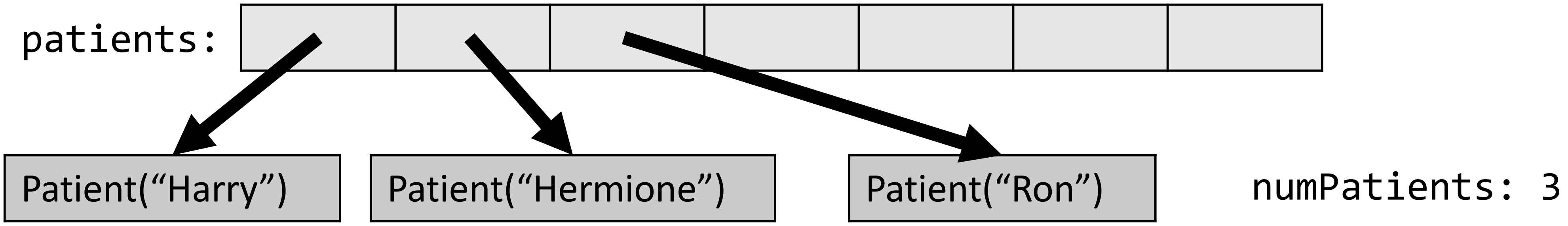
What a PatientDatabase needs to be able to do

An example of using a PatientDatabase

```
class PatientDatabase {
  private Patient[] patients;
  private int numPatients;


  PatientDatabase(int maxPatients) {
    patients = new  Patient[maxPatients];
    numPatients = 0;
  }

  // Register a new Patient in the database
  // return false if out of space
  boolean registerNewPatient(String name) { ... }

  // Print all patient names in alphabetical order
  void printNamesAlphabetically() { ... }

}
```

Let's ***implement*** the PatientDatabase with an array, which we'll keep sorted

EXAMPLE



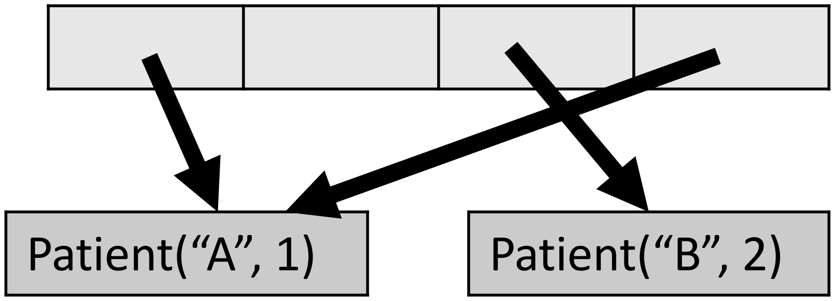
Peer instruction

https://b.socrative.com/login/student/
CS2230A

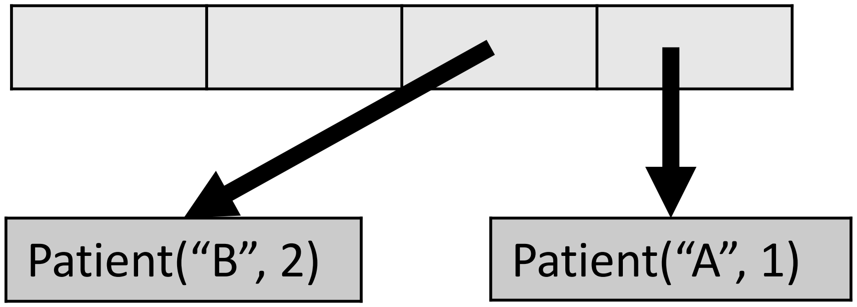
```
Patient[] arr = new Patient[4];
Patient pa = new Patient("A", 1);
Patient pb = new Patient("B", 2);
arr[0] = pa;
arr[2] = pb;
arr[3] = pa;
```

What is the result of running this code?

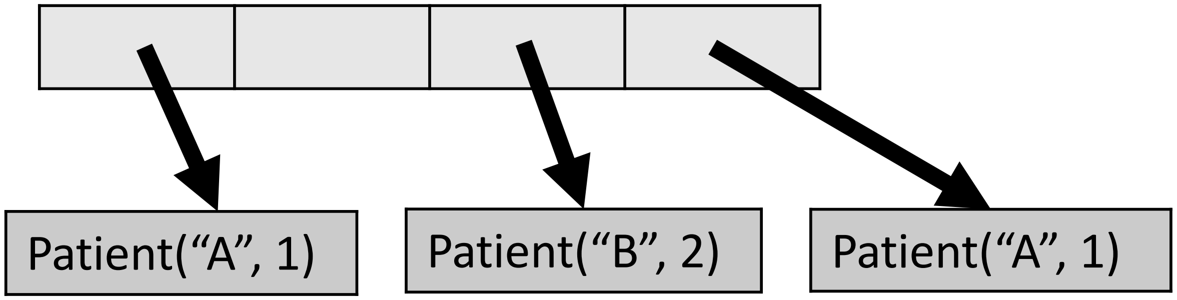
(A)



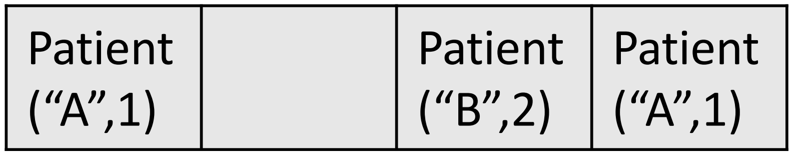
(B)



(C)



(D)

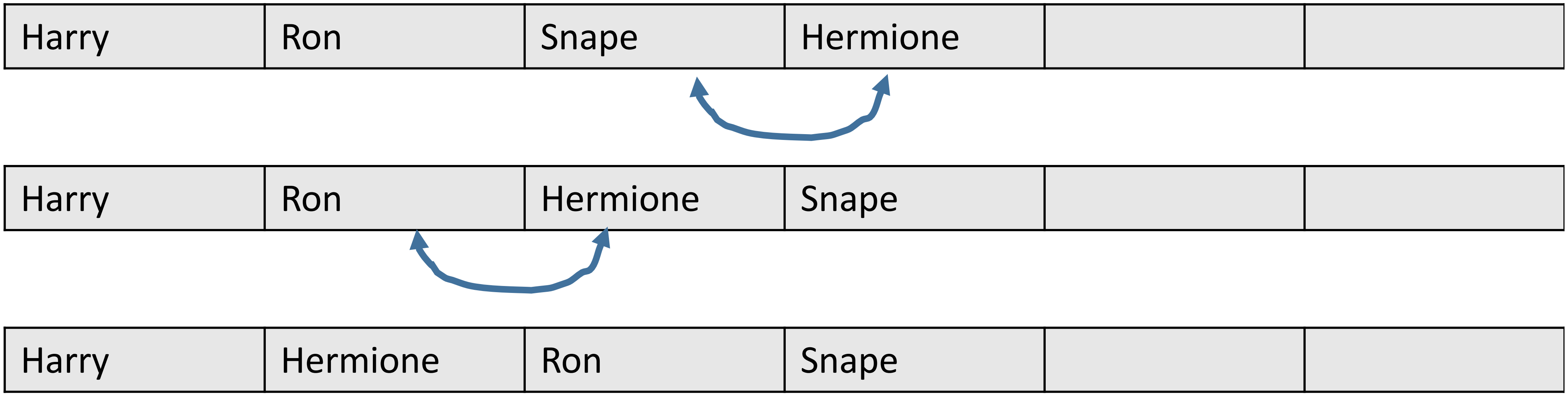


```java
// Register a new Patient in the database
boolean registerNewPatient(String name) { ... }
```

algorithm: insert new element at the the end, then swap until it is in the right place

Harry	Ron	Snape			

```java
registerNewPatient("Hermione")
```



```java
// Register a new Patient in the database (if we have space)
boolean registerNewPatient(String name) {
  if (numPatients == patients.length) return false;

  // since they haven't been measured we will give height=0
  Patient newp = new Patient(name, 0);

  // start with the new patient at the end of the list
  patients[numPatients] = newp;

  numPatients+=1;

  // keep swapping the patient with the previous patient
  // until it is in alphabetical order
  int i = numPatients-1;
  while (i > 0 &&
         patients[i].name.compareTo(patients[i-1].name) < 0) {

     swapPatients(i, i-1);
     i--;
   }

   return true;
}
```

"Alice".compareTo("Bob") → -1
"Bob".compareTo("Alice") → 1

we'll get to swapPatients() next

```java
class PatientDatabase {
  private Patient[] patients;


  private void swapPatients(int a, int b) {
    Patient pa = patients[a];
    Patient pb = patients[b];
    patients[a] = pb;
    patients[b] = pa;
  }


  boolean registerNewPatient(String name) {

    while ( ) {

      swapPatients(i, i-1);

    }
  }
}
```

```java
class PatientDatabase {
  private Patient[] patients;


  private void swapPatients(int a, int b) {
    Patient pa = patients[a];
    Patient pb = patients[b];
    patients[a] = pb;
    patients[b] = pa;
  }


  boolean registerNewPatient(String name) {

    while ( ) {

      swapPatients(i, i-1);

    }
  }
}
```

`swapPatients()` wouldn't make sense to outsiders!

- it is just an *implementation detail* used by `registerNewPatient()`
- `PatientDatabase` could have been implemented with something other than an array sorted by names

Peer instruction

https://b.socrative.com/login/student/
CS2230A

Making `patients` and `swapPatients` private is *most* an example of which object-oriented design principle?

A) Abstraction
B) Encapsulation
C) Modularity

(page 61 of GTG)

```
class PatientDatabase {
  private Patient[] patients;


  private void swapPatients(int a, int b) {

  }


  boolean registerNewPatient(String name) {

    while ( ) {

      swapPatients(i, i-1);

    }
  }
}
```

Implementation of the other method

```
// Print all patient names in alphabetical order
void printNamesAlphabetically() {
  for (int i=0; i<numPatients; i++) {
    System.out.println(patients[i].name);
  }
}
```

(see the whole PatientDatabase class in PatientDatabase.java, and run the program for yourself)

https://b.socrative.com/login/student/
CS2230A

Think-pair-share (peer instruction)

Why is it important to distinguish between the *specification* and the *implementation* of a class? (short answer)

Today’s big ideas

- When we want an array of objects, we store their *references* in the array
- It is important to distinguish between the specification and implementation of a class
- **public** and **private** control access to fields and methods