Lecture 4

Classes and Objects

Review

Solutions 1

```
public static int getMinIndex(int[] values) {

      int minValue = Integer.MAX_VALUE;
      int minIndex = -1;

      for(int i=0; i<values.length; i++)
            if (values[i] < minValue) {
                  minValue = values[i];
                  minIndex = i;
            }

      return minIndex;
}
```

Solutions 2

```
public static int getSecondMinIndex(int[] values) {
      int secondIdx = -1;
      int minIdx= getMinIndex(values);

      for(int i=0; i<values.length; i++) {
            if (i == minIdx)
                  continue;
            if (secondIdx == -1 ||
               values[i] < values[secondIdx])
                  secondIdx = i;
      }
      return secondIdx;
}
```

- What happens if values = {0}? values = {0, 0}? values = {0,1}?

Popular Issues 1

- Array Index vs Array Value

```
int[] values = {99, 100, 101};
System.out.println(values[0] );   // 99
```

Values	99	100	101
Indexes	0	1	2

Popular Issues 2

- Curly braces { … } after if/else, for/while

```
for (int i = 0; i < 5; i++)
    System.out.println("Hi");
    System.out.println("Bye");
```

- What does this print?

Popular Issues 3

- Variable initialization

```
int getMinValue(int[] vals) {
  int min = 0;
  for (int i = 0; i < vals.length; i++) {
   if (vals[i] < min) {
         min = vals[i]
   }
  }
  return min;
}
```

- What if `vals = {1,2,3}`? ← Problem?
- Set `min = Integer.MAX_VALUE` or `vals[0]`

Popular Issues 4

- Variable Initialization – secondMinIndex

```
int minIdx = getMin(vals)
int secondIdx = 0;
for (int i = 0; i < vals.length; i++) {
    if (i == minIdx) continue;
    if (vals[i] < vals[secondIdx])
          secondIdx = i;
}
```

- What if vals = {0, 1, 2}?
- See solutions

Popular Issues 5

Defining a method inside a method

```
public static void main(String[] arguments) {
    public static void foobar () {
    }
}
```

Debugging Notes 1

- System.out.println is your friend

```
for ( int i=0; i< vals.length; i++) {
  if ( vals[i] < minVal) {
      System.out.println("cur min: " + minVal);
      System.out.println("new min: " + vals[i]);
      minVal = vals[i];
  }
}
```

Debugging Notes 2

- Formatting
- Ctrl-shift-f is your friend

```
for (int i = 0; i < vals.length; i++) {
    if (vals[i] < vals[minIdx]) {
minIdx=i;}
return minIdx;}
```

- Is there a bug? Who knows! Hard to read

Today's Topics

Object oriented programming

Defining Classes

Using Classes

Today's Topics

Object oriented programming

Defining Classes

Using Classes

References vs Values

Static types and methods

Whew!

That's a lot!

Object Oriented Programming

- Represent the real world

Baby

Object Oriented Programming

- Represent the real world

Baby

Name
Sex
Weight
Decibels
poops so far

Object Oriented Programming

- Objects group together
  - Primitives (int, double, char, etc..)
  - Objects (String, etc…)

Baby

```
String name
boolean isMale
double weight
double decibels
int numPoops
```

Why use classes?

- Why not just primitives?

```
// little baby alex
String nameAlex;
double weightAlex;
// little baby david
String nameDavid;
double weightDavid;
```

Why use classes?

- Why not just primitives?

```
// little baby alex
String nameAlex;
double weightAlex;
// little baby david
String nameDavid;
double weightDavid;
// little baby david
String nameDavid2;
double weightDavid2;
```

David2?
Terrible ☹

Why use classes?

- Why not just primitives?

```
// little baby alex
String nameAlex;
double weightAlex;
// little baby david
String nameDavid;
double weightDavid;
// little baby david
String nameDavid2;
double weightDavid2;
```

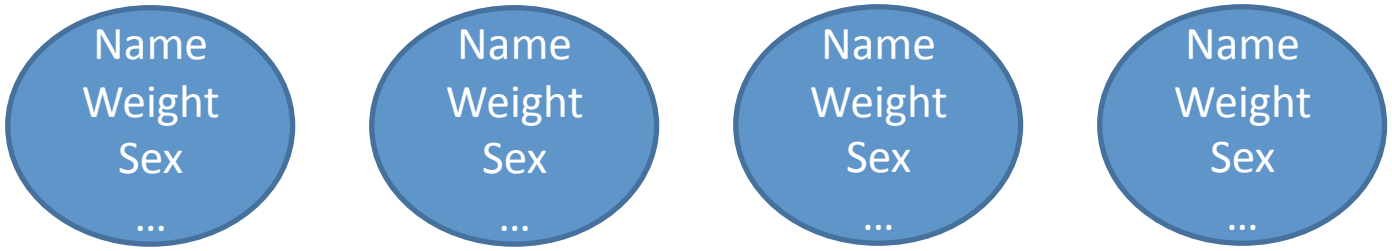
David2?
Terrible ☹

500 Babies? That Sucks!

Why use classes?

Baby1

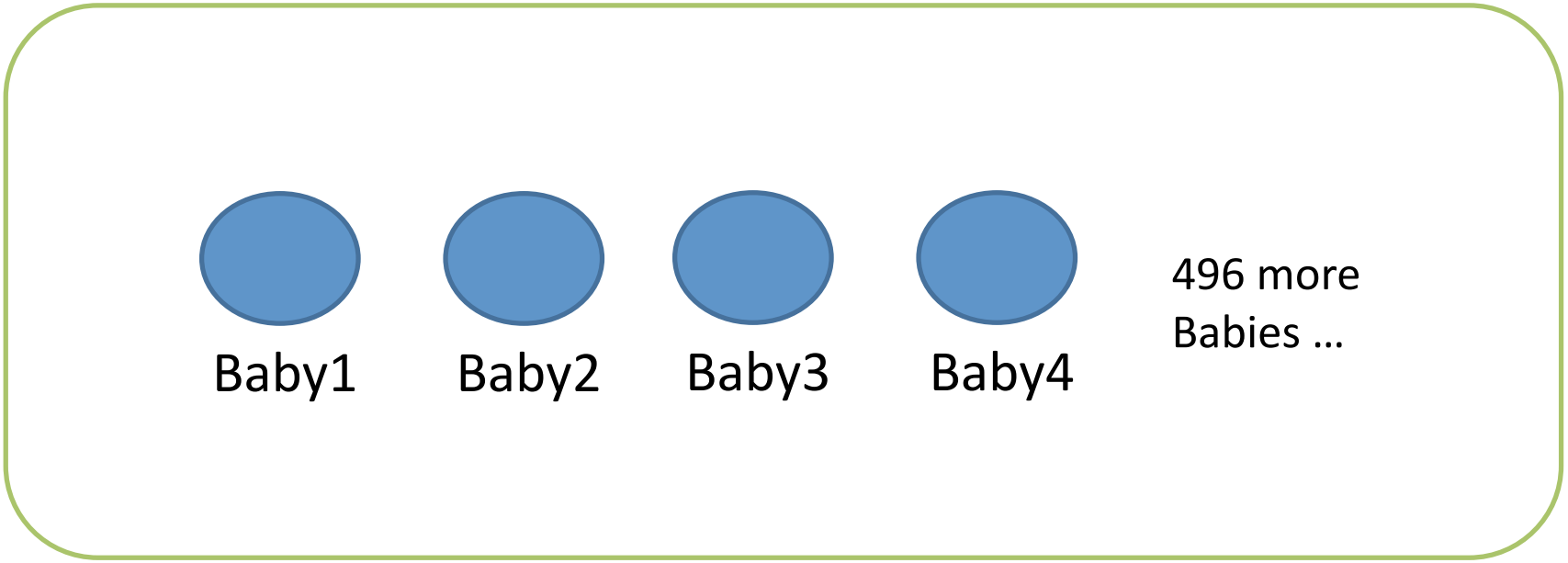
Why use classes?



Baby1 Baby2 Baby3 Baby4

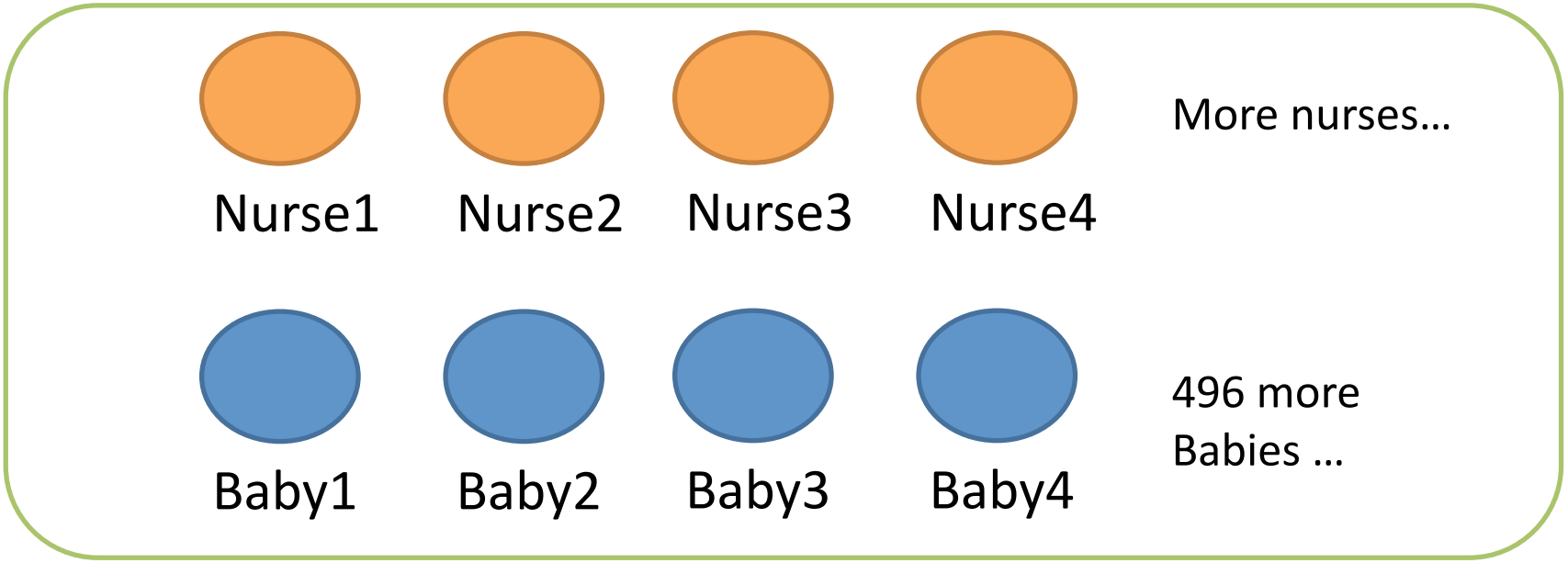
496 more Babies ...

Why use classes?



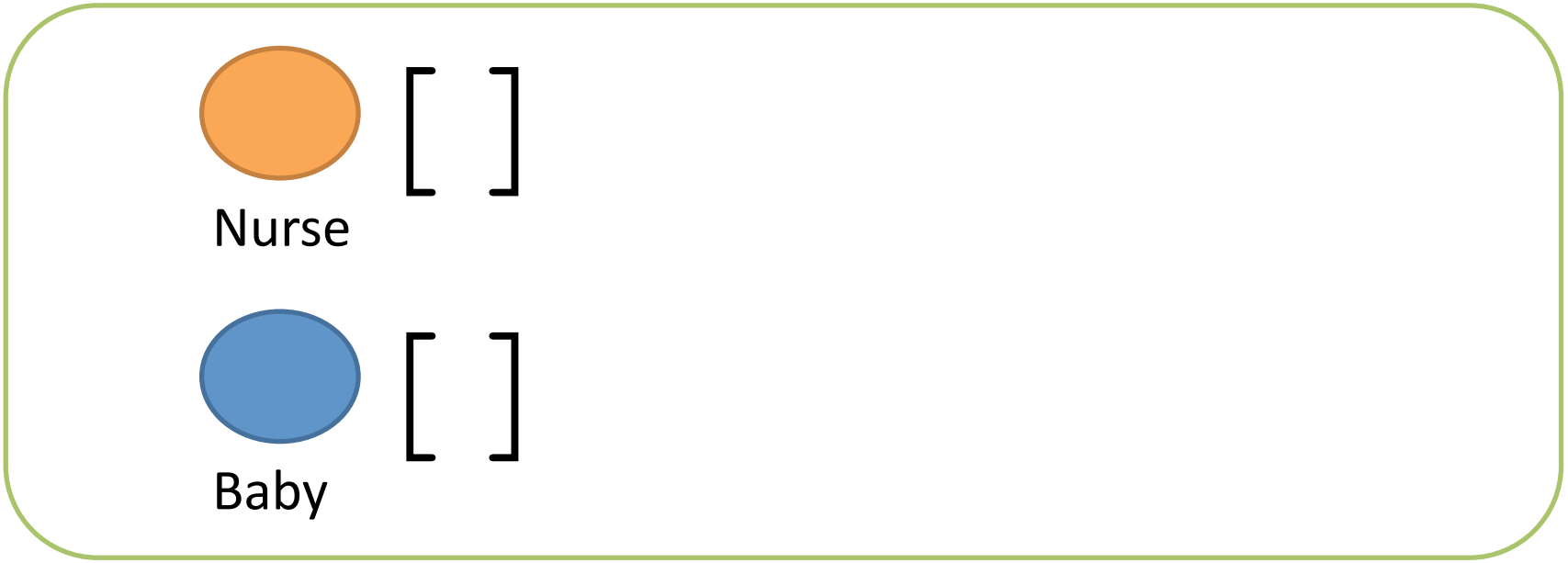
Nursery

Why use classes?



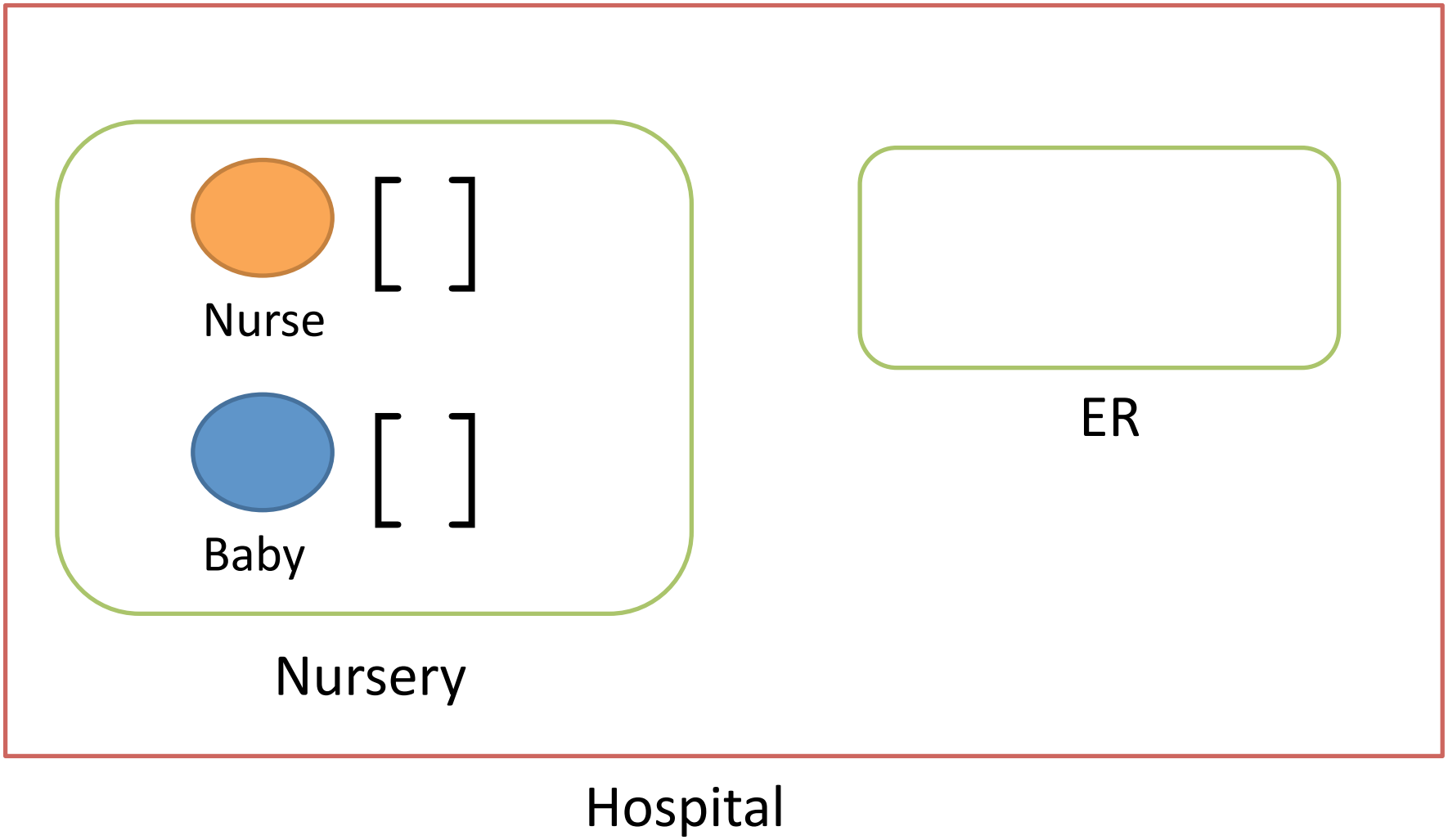
Nursery

Why use classes?



Nursery

Why use classes?



Defining classes

Overview

```
public class Baby {
    String name;
    boolean isMale;
    double weight;
    double decibels;
    int numPoops = 0;

    void poop() {
      numPoops += 1;
      System.out.println("Dear mother, "+
           "I have pooped.  Ready the diaper.");
    }
}
```

Class

Definition

Overview

```
Baby myBaby = new Baby();
```

Class

Instance

Let’s declare a baby!

```
public class Baby {

}
```

Let’s declare a baby!

```
public class Baby {

    fields

    methods


}
```

Let’s declare a baby!

```
public class Baby {
```

fields

methods

public static void main()

```
}
```

Let's declare a baby!

```
public class Baby {
```

Class names are capitalized

fields

methods

Class name = File name

public static void main()

main() = can execute

```
}
```

Fields

```
public class Baby {

    fields



}
```

Fields

```
public class Baby {

    TYPE var_name;
    TYPE var_name = some_value;



}
```

Fields

```
public class Baby {
   String name;
   double weight = 5.0;
   boolean isMale;



}
```

Quiz: Add field for siblings

```
public class Baby {
   String name;
   double weight = 5.0;
   boolean isMale;
   XXXXX  YYYYY;

}
```

Quiz: Add field for siblings

```
public class Baby {
    String name;
    double weight = 5.0;
    boolean isMale;
    Baby[] siblings;

}
```

Baby class so far

```
public class Baby {
    String name;
    double weight = 5.0;
    boolean isMale;
    Baby[] siblings;
}
```

Methods

```
public class Baby {

}
```

methods

Baby methods

```
public class Baby {

    String name = “Slim Shady”;

    void sayHi() {
          System.out.println(
          “Hi, my name is.. “ + name);
    }
}
```

Baby methods

```
public class Baby {

    double weight = 5.0;

    void eat(double foodWeight) {
          if (foodWeight >= 0) {
                weight = weight + foodWeight;
          }
    }
}
```

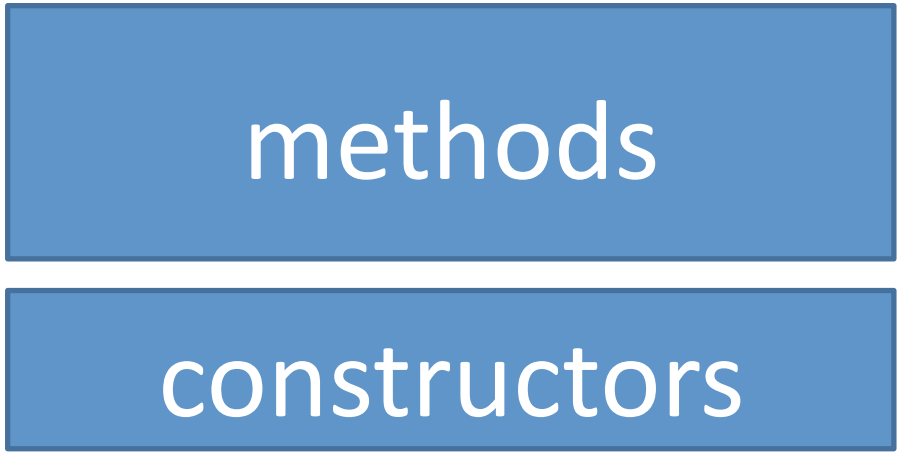
Baby class so far

```
public class Baby {
    String name;
    double weight = 5.0;
    boolean isMale;
    Baby[] siblings;

    void sayHi() { … }
    void eat(double foodWeight) { … }
}
```

Constructors

```
public class Baby {
```




```
}
```

Ok, let’s make this baby!

```
Baby ourBaby = new Baby();
```

Ok, let’s make this baby!

```
Baby ourBaby = new Baby(name, sex);
```

Babies start out with a name and sex!

Constructors

```
public class CLASSNAME{
    CLASSNAME ( ) {
    }

    CLASSNAME ([ARGUMENTS]) {
    }
}


CLASSNAME obj1 = new CLASSNAME();
CLASSNAME obj2 = new CLASSNAME([ARGUMENTS])
```

Constructors

- Constructor name == the class name
- No return type – never returns anything
- Usually initialize fields
- All classes need at least one constructor
  - If you don't write one, defaults to

```
CLASSNAME () {
}
```

Baby constructor

```
public class Baby {
    String name;
    boolean isMale;
    Baby(String myname, boolean bMale){
          name = myname;
          isMale = bMale;
    }
}
```

Baby class so far

```
public class Baby {
    String name;
    double weight = 5.0;
    boolean isMale;
    Baby[] siblings;

    void sayHi() { … }
    void eat(double foodWeight) { … }
    Baby(String myname, boolean bMale){ … }
}
```

Using classes

Classes and Instances

```
// class Definition
public class Baby {…}


// class Instances
Baby bart = new Baby(“Bart Simpson”, true);
Baby lisa = new Baby(“Lisa Simpson”, false);
```

Accessing fields

- Object.FIELDNAME

```
Baby bart = new Baby(“Bart Simpson”, true)
System.out.println(bart.name);
System.out.println(bart.weight);
```

Calling Methods

- Object.METHODNAME([ARGUMENTS])

```
Baby bart = new Baby(“Bart Simpson”, true)
bart.sayHi();   // “Hi, my name is Bart Simpson”
bart.eat(1);
```

References vs Values

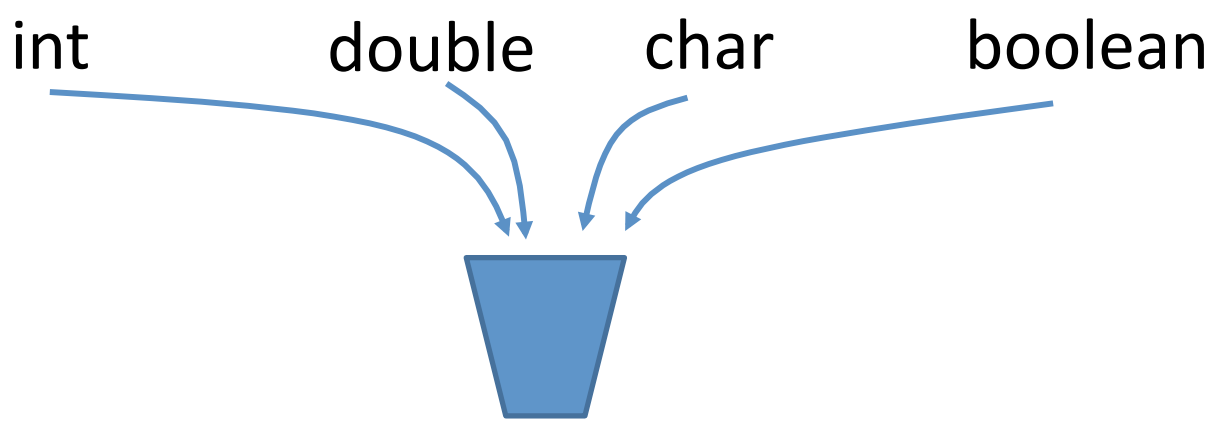
Primitives vs References

- Primitive types are basic java types
  - int, long, double, boolean, char, short, byte, float
  - The actual values are stored in the variable

- Reference types are arrays and objects
  - String, int[], Baby, …

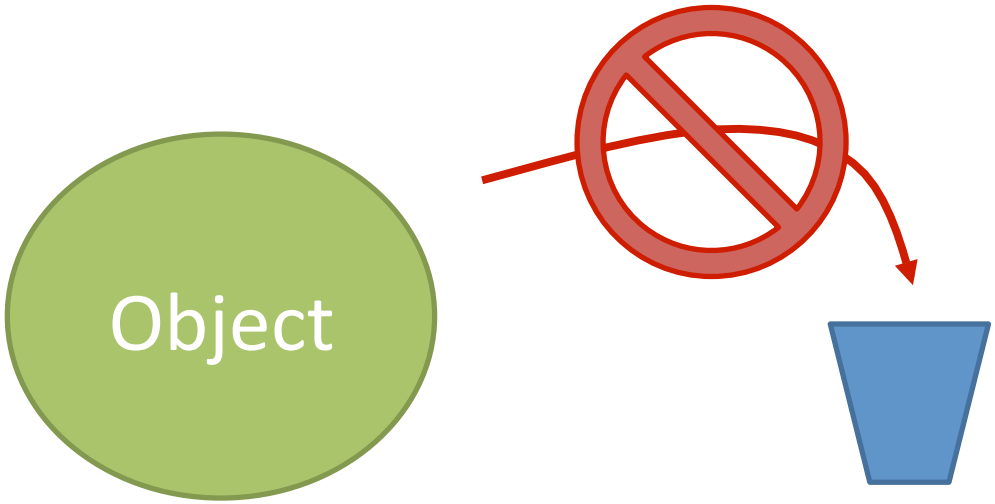
How java stores primitives

- Variables are like fixed size cups
- Primitives are small enough that they just fit into the cup



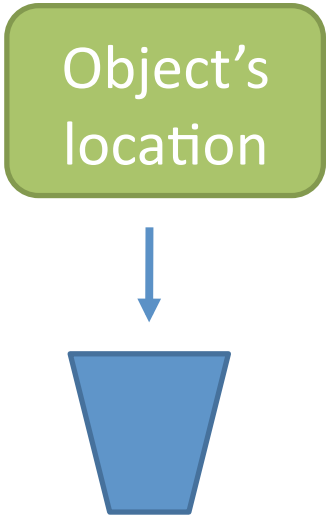
How java stores objects

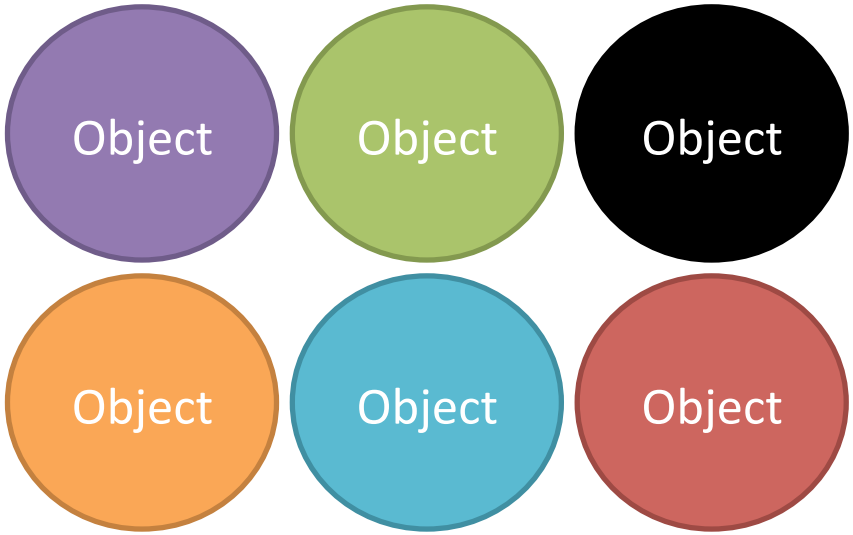
- Objects are too big to fit in a variable
  - Stored somewhere else
  - Variable stores a number that locates the object



How java stores objects

- Objects are too big to fit in a variable
  - Stored somewhere else
  - Variable stores a number that locates the object





References

- The object's location is called a reference
- `==` compares the references

```
Baby shiloh1 = new Baby("shiloh");
Baby shiloh2 = new Baby("shiloh");
```

Does `shiloh1 == shiloh2`?

References

- The object's location is called a reference
- `==` compares the references

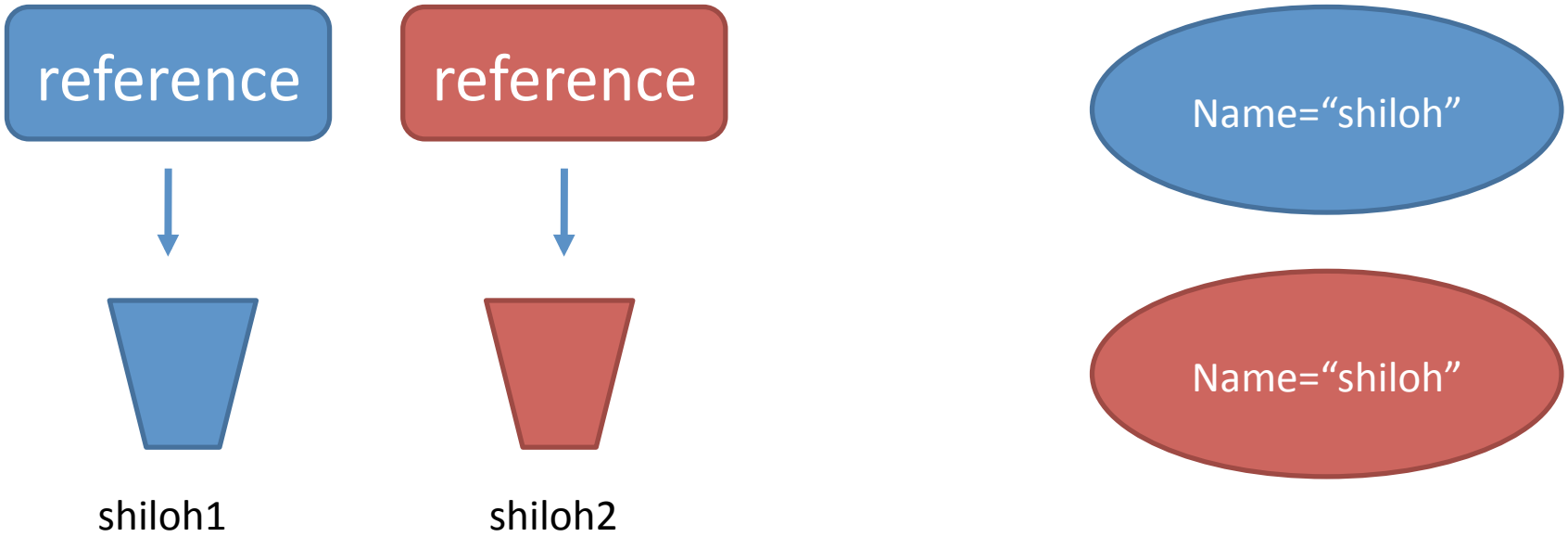
```
Baby shiloh1 = new Baby("shiloh");
Baby shiloh2 = new Baby("shiloh");
```

Does `shiloh1 == shiloh2`?

no

References

```
Baby shiloh1 = new Baby("shiloh");
Baby shiloh2 = new Baby("shiloh");
```

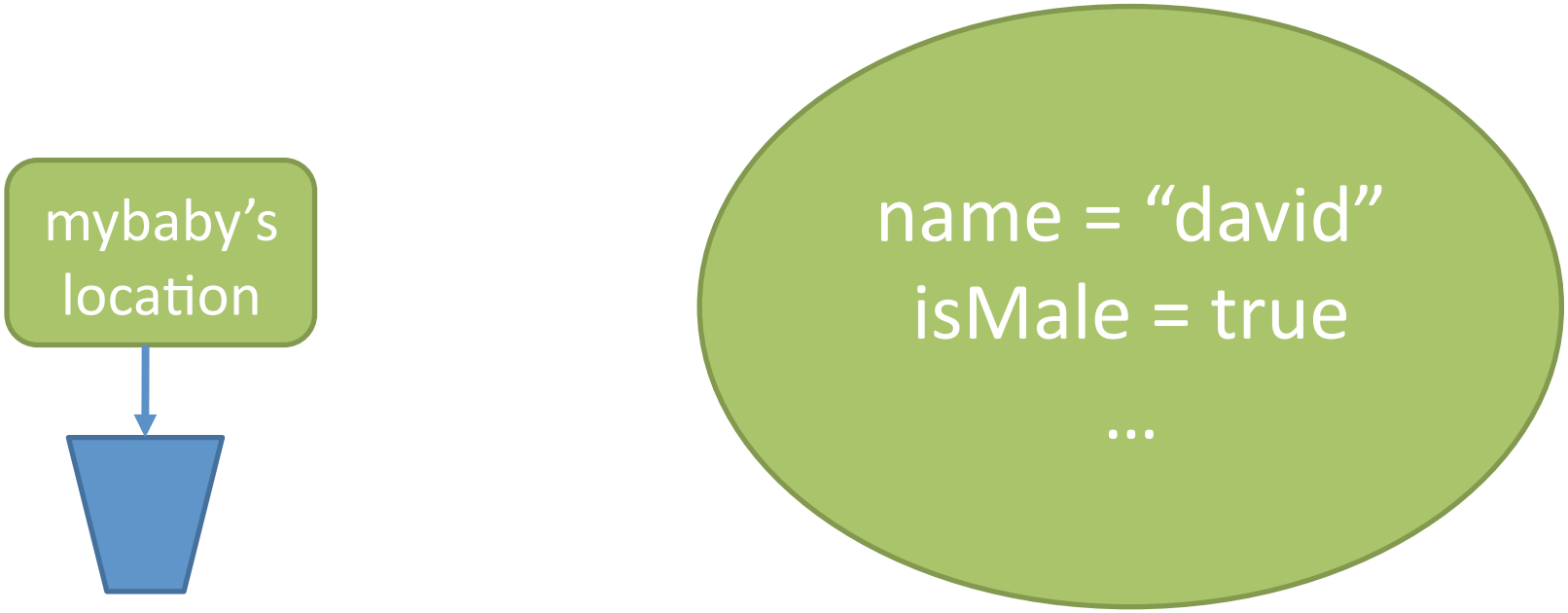


References

```
Baby mybaby = new Baby(“davy”, true)
mybaby.name = “david”
```

References

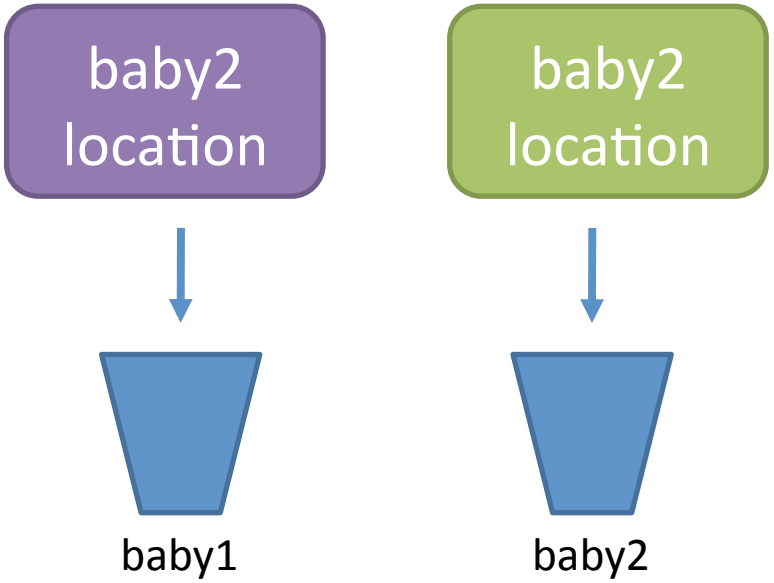
```
Baby mybaby = new Baby(“davy”, true)
mybaby.name = “david”
```

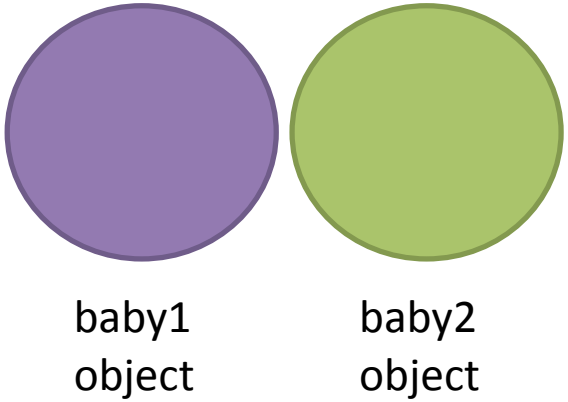


References

- Using = updates the reference.

```
baby1 = baby2;
```

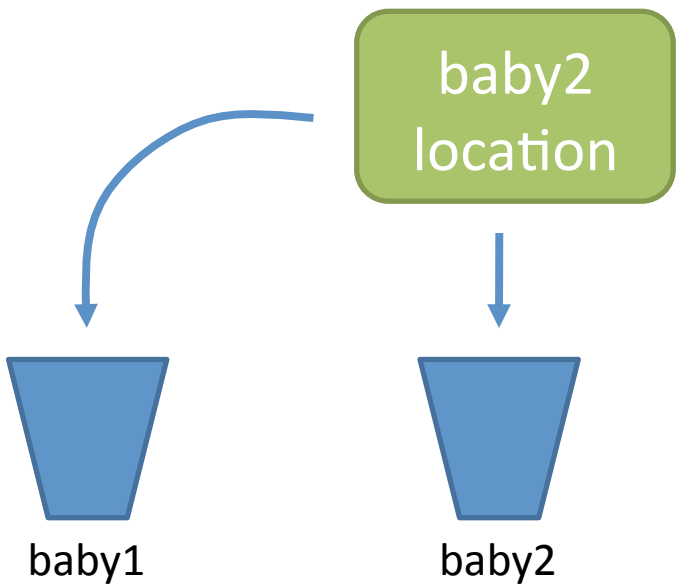


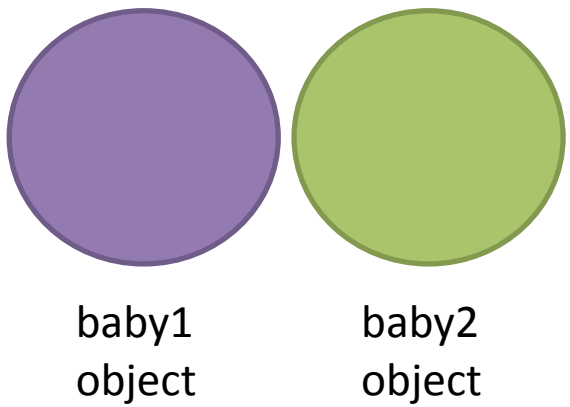


References

- Using = updates the reference.

```
baby1 = baby2;
```





References

- using [ ] or •
  - Follows the reference to the object
  - May modify the object, but never the reference
- Imagine
  - Following directions to a house
  - Moving the furniture around
- Analogous to
  - Following the reference to an object
  - Changing fields in the object

Methods and references

```
void doSomething(int x, int[] ys, Baby b) {
    x = 99;
    ys[0] = 99;
    b.name = “99”;
}
...

int i = 0;
int[] j = {0};
Baby k = new Baby(“50”, true);
doSomething(i, j, k);
```

i=? j=? k=?