IT ENTREPRENEURSHIP

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]
    }
   }
}
```

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

Popular Issues 4

Defining a method inside a method

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

Debugging Notes 1

- Use System.out.println throughout your code to see what it's doing

```
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

References vs Values

Static types and methods

Object oriented programming

- Represent the real world

Baby

Object oriented programming

- Represent the real world

Baby

Name
Gender
Weight
Decibels

Object Oriented Programming

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

Baby

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

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

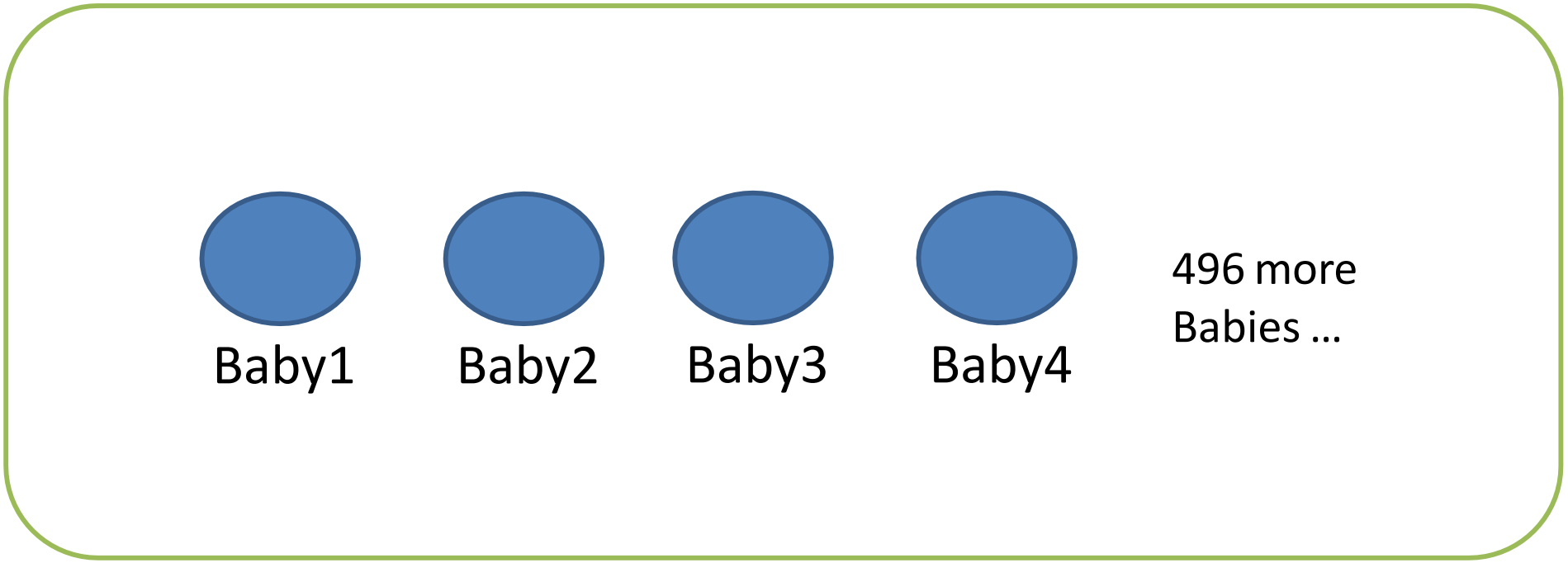

500 Babies? That Sucks!

Why use classes?

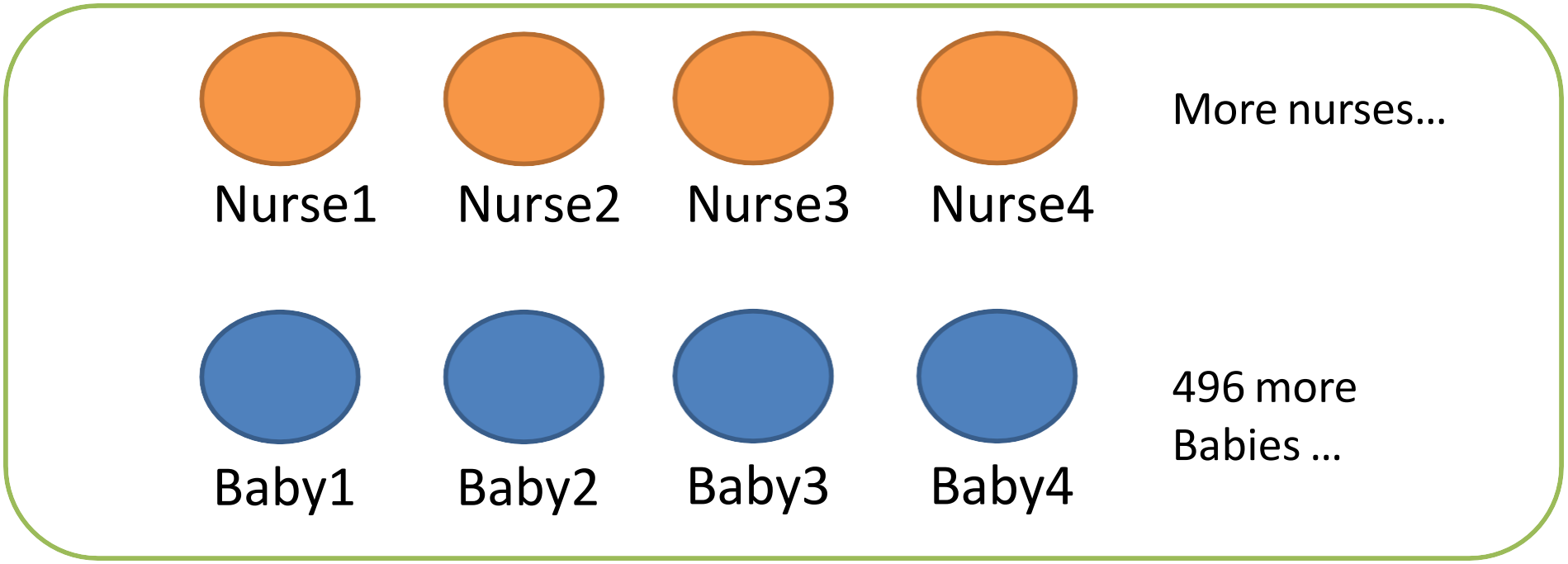
Baby1

Why use classes?

Why use classes?

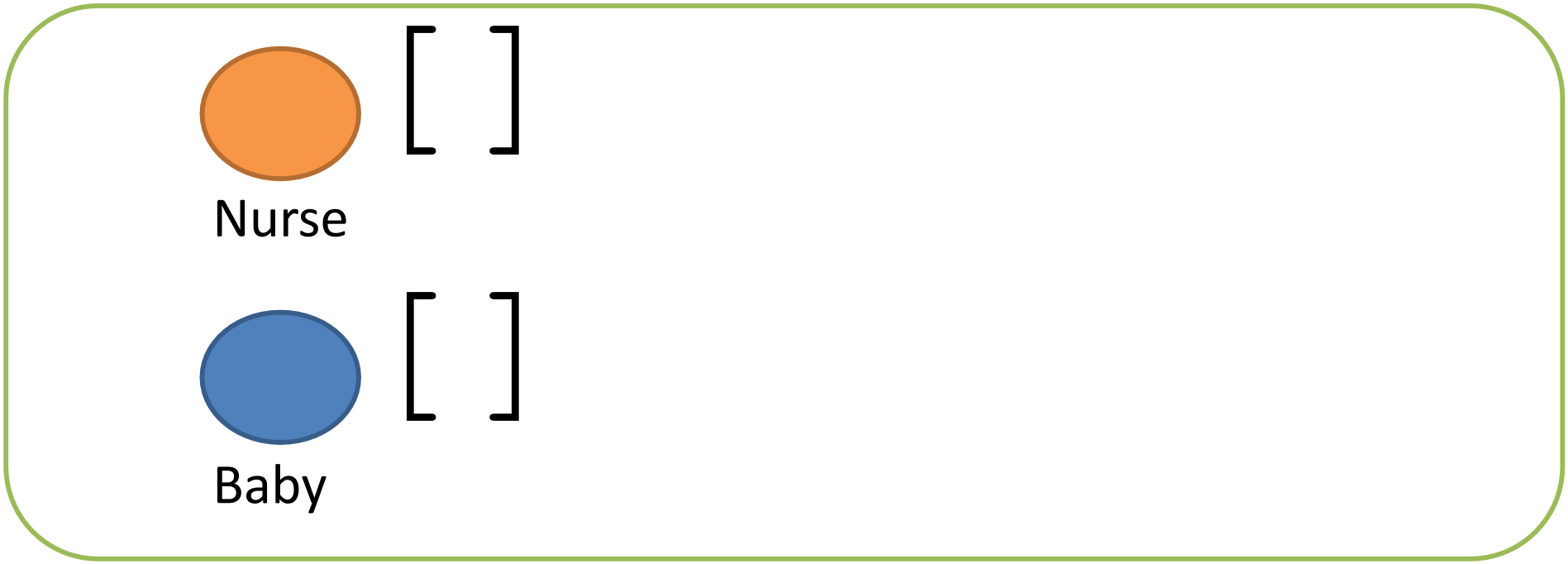


Why use classes?

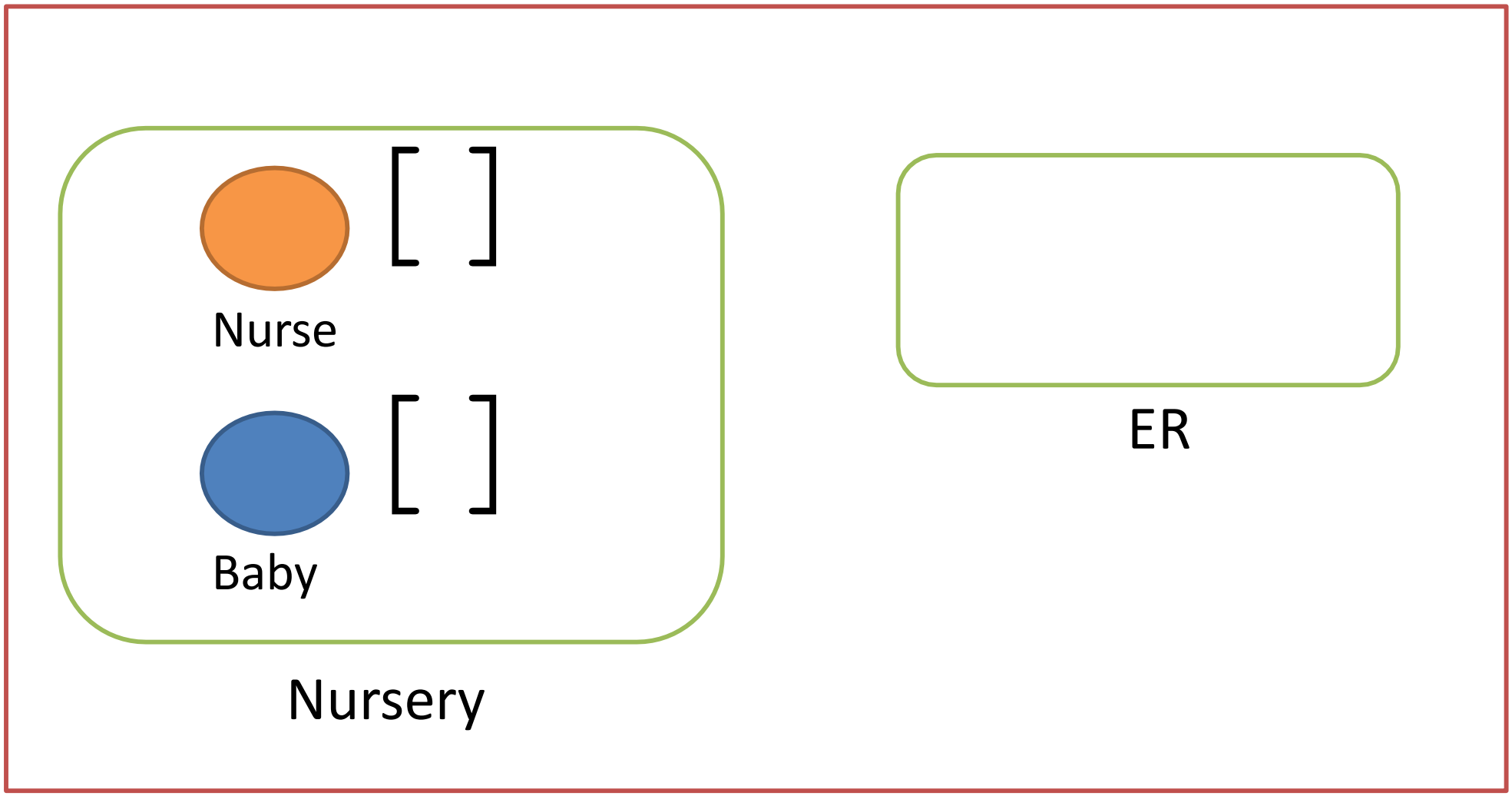


Nursery

Why use classes?



Why use classes?



Defining classes

Class - overview

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




}
```

Class
Definition

Class - 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

Note

- Class names are Capitalized
- 1 Class = 1 file
- Having a `main` method means the class can be run

Baby fields

```
public class Baby {

    TYPE var_name;
    TYPE var_name = some_value;



}
```

Baby fields

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



}
```

Baby Siblings?

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


}
```

Baby Siblings?

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

}
```

Ok, let's make this baby!

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

But what about it's name? it's gender?

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 maleBaby){
          name = myname;
          isMale = maleBaby;
    }
}
```

Baby methods

```
public class Baby {
    String name = “Slim Shady”;
    ...
    void sayHi() {
        System.out.println(
        “Hi, my name is.. “  + name);
    }
}
```

Baby methods

```
public class Baby {
    String weight  = 5.0;

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

}
```

Baby class

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


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

Using classes

Classes and Instances

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


// class Instances
Baby shiloh = new Baby(“Shiloh Jolie-Pitt”, true);
Baby knox   = new Baby(“Knox Jolie-Pitt”,   true);
```

Accessing fields

- Object.FIELDNAME

```
Baby shiloh = new Baby("Shiloh Jolie-Pitt",
                       true)
System.out.println(shiloh.name);
System.out.println(shiloh.weight);
```

Calling Methods

- Object.METHODNAME([ARGUMENTS])

```
Baby shiloh = new Baby("Shiloh Jolie-Pitt",
                       true)
shiloh.sayHi();      // "Hi, my name is ..."
shiloh.eat(1);
```

References vs Values

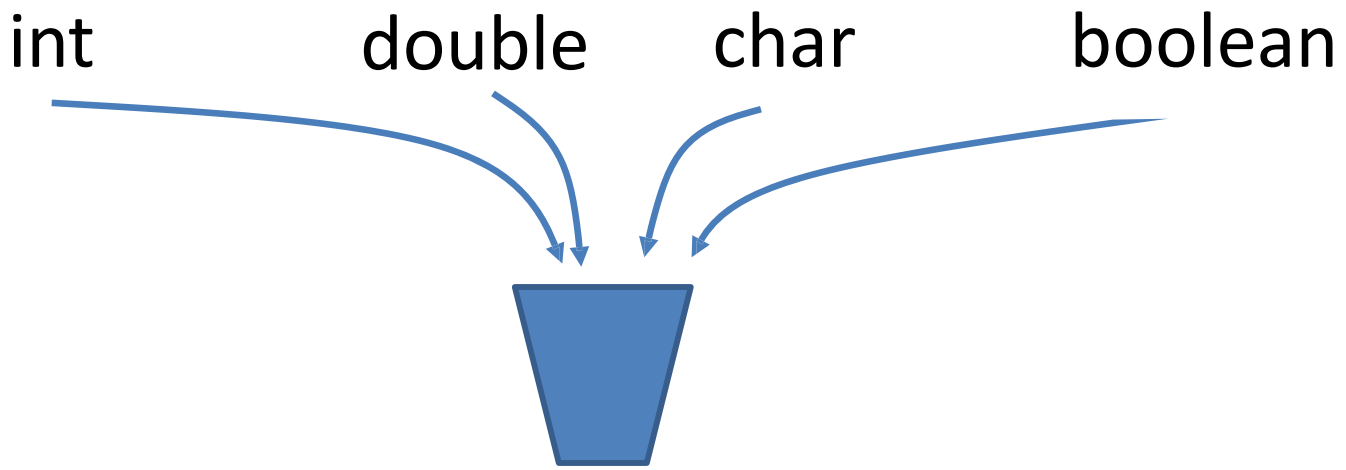
Primitives vs References

- 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, …

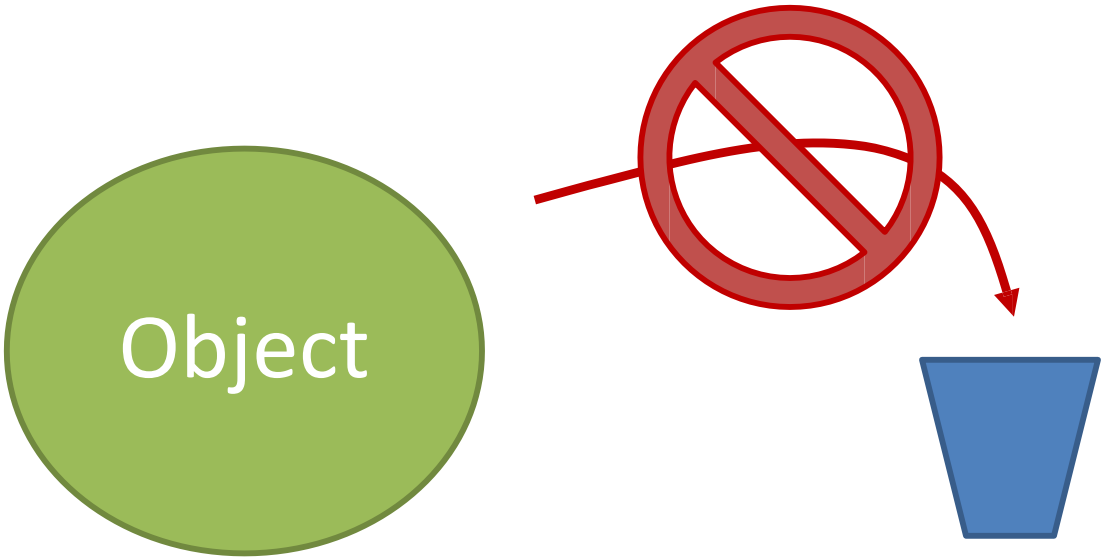
How java stores primitives

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



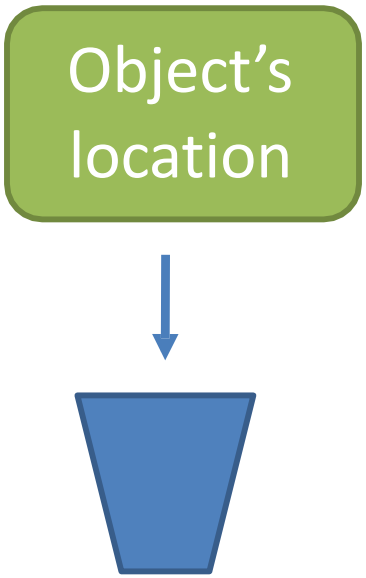
How java stores objects

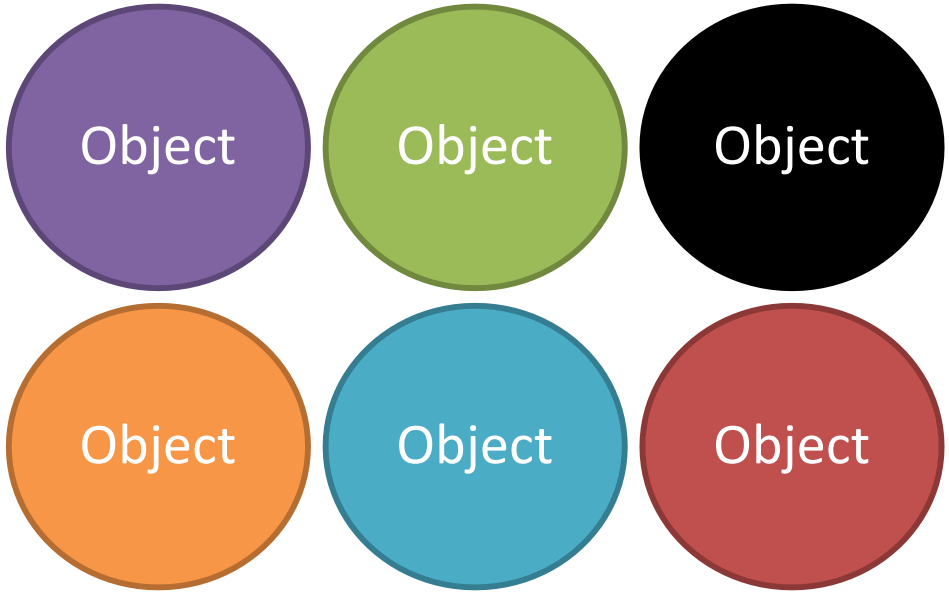
- Objects are too big to fit in a variable
  - Stored somewhere else
  - Variable stores a number that locates the object



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

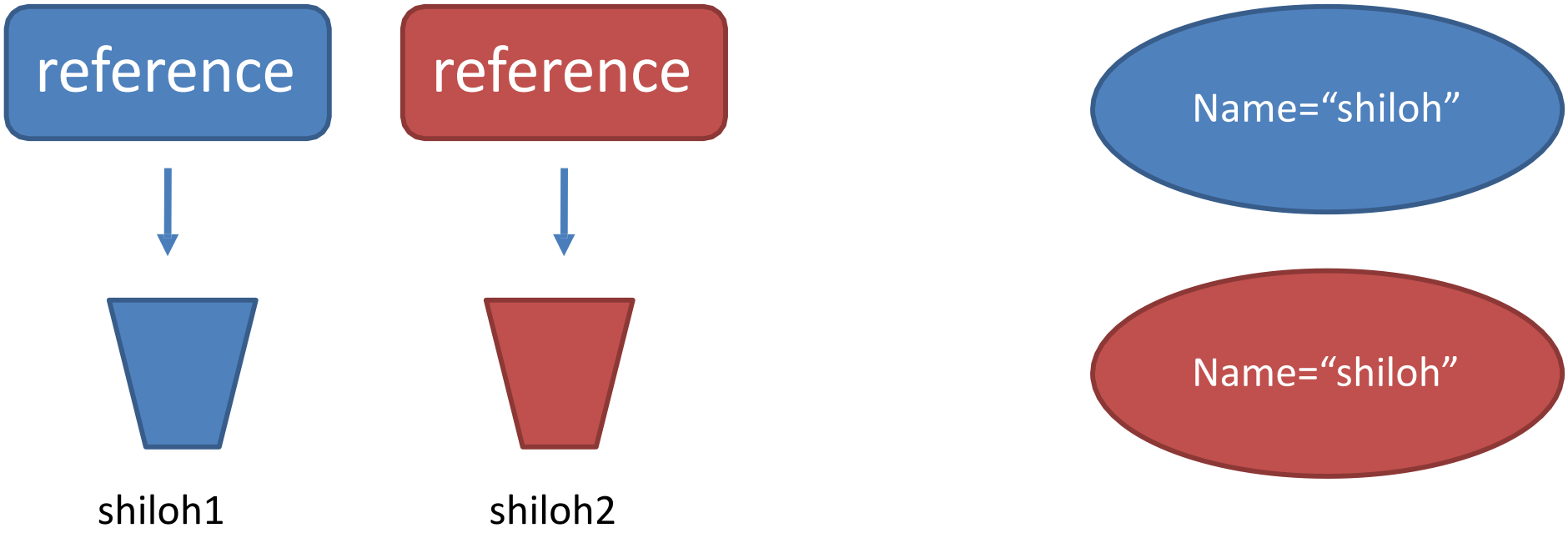
```
Baby shiloh1 = new Baby("shiloh");
Baby shiloh2 = new Baby("shiloh");
```

Does `shiloh1 == shiloh2`?

no

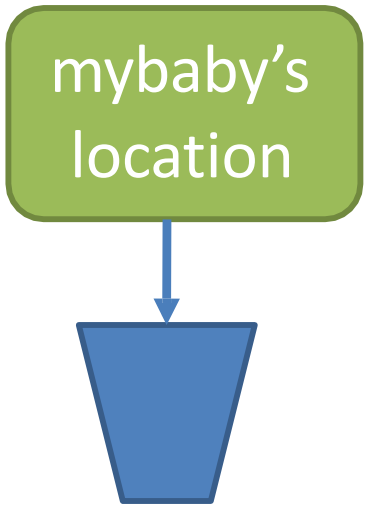
References

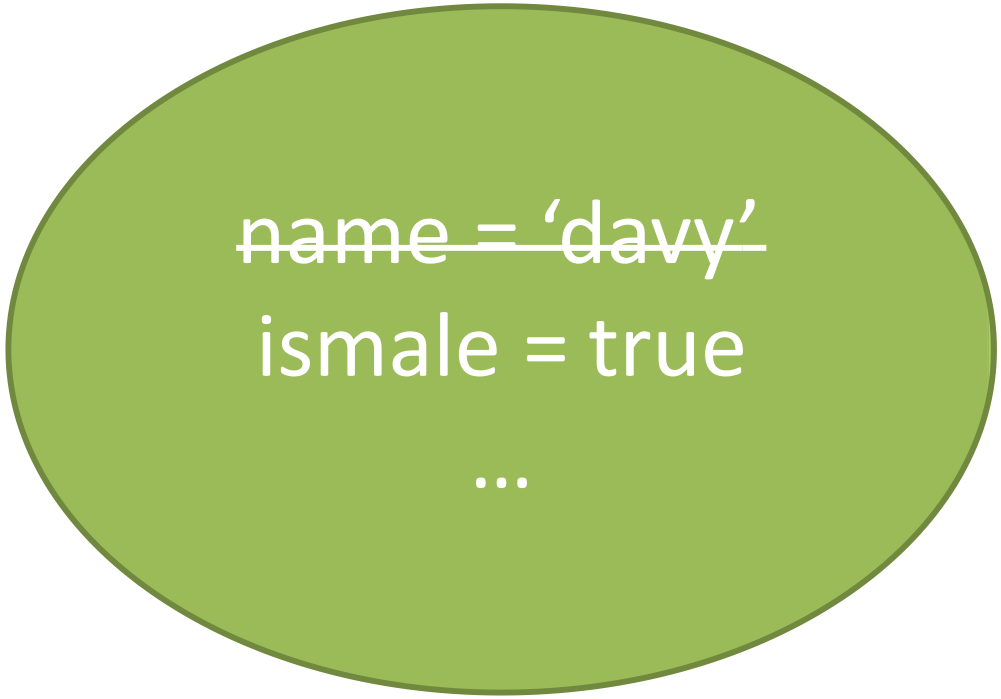
```
Baby shiloh1 = new Baby("shiloh");
Baby shiloh2 = new Baby("shiloh");
```



References

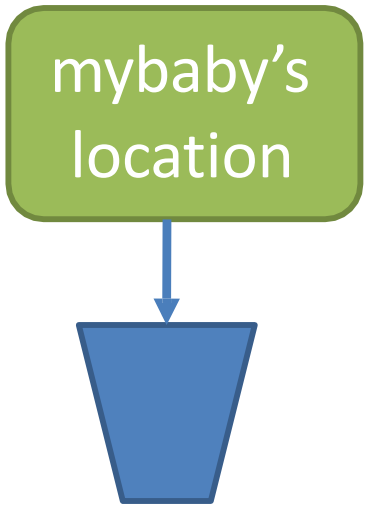
```
Baby mybaby = new Baby("davy", true)
mybaby.name = "david"
```

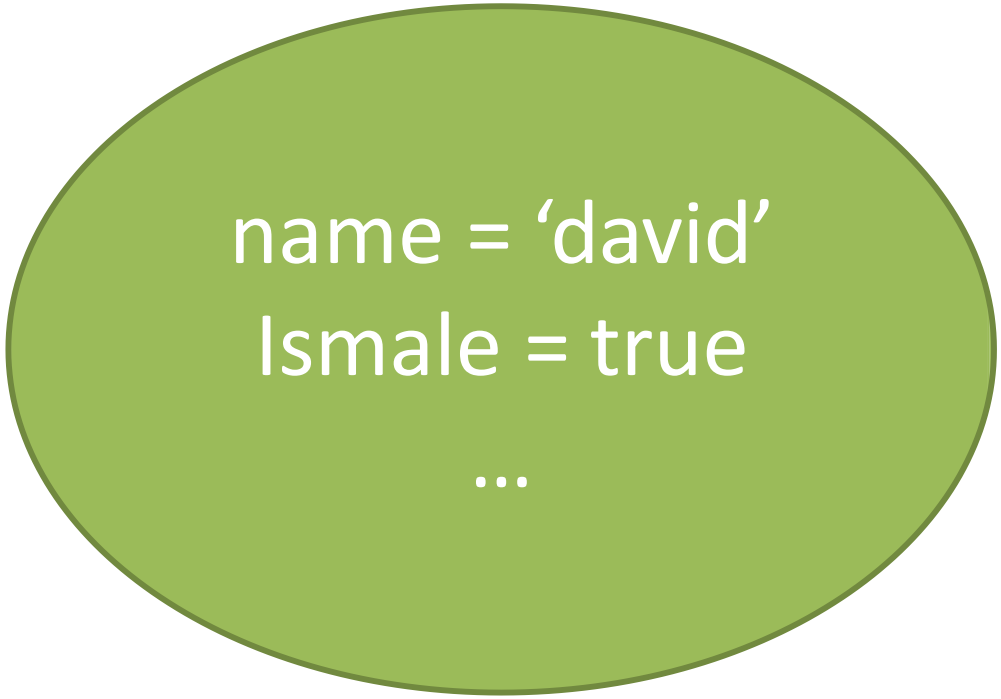




References

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

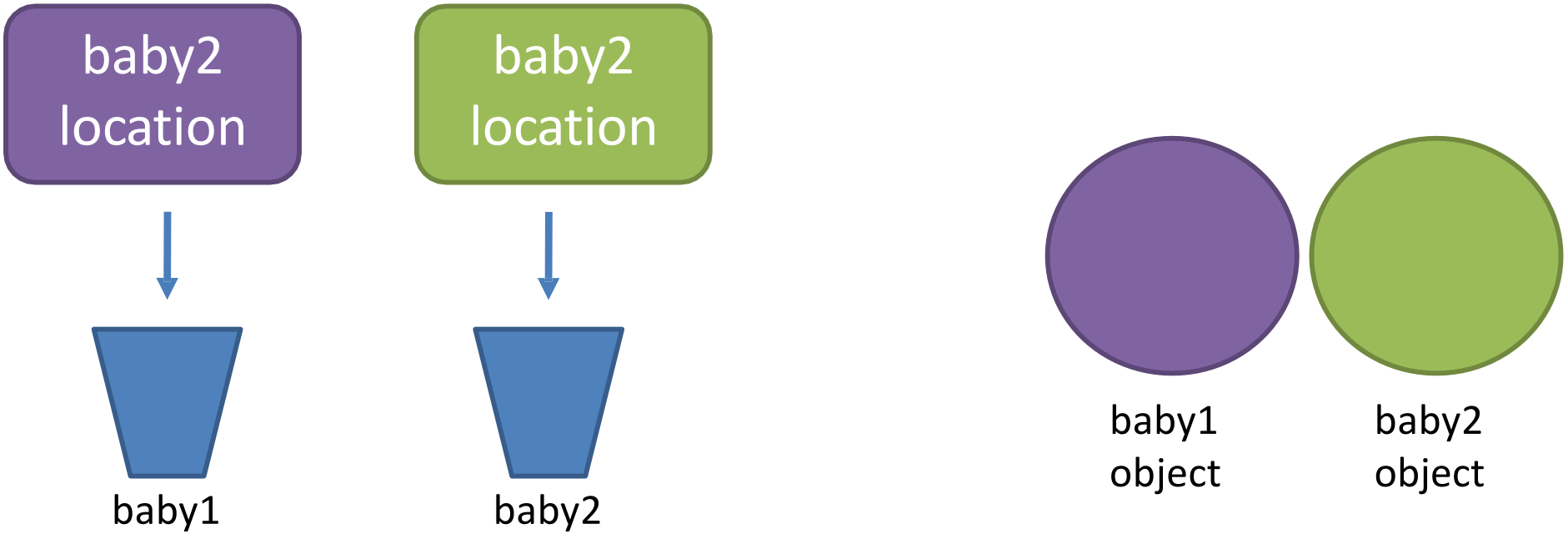




References

- Using = updates the reference.

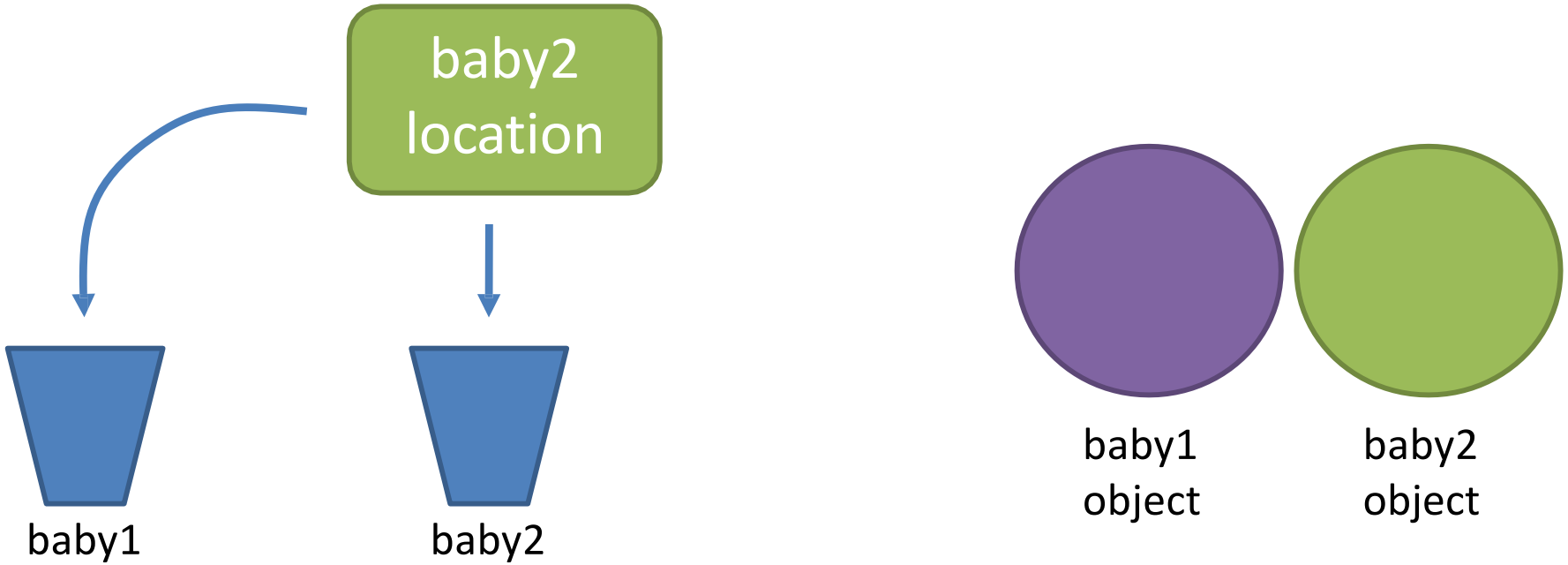
```
baby1 = baby2
```



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=?

static types and methods

static

- Applies to fields and methods
- Means the field/method
  - Is defined for the class declaration,
  - Is not unique for each instance

static

```
public class Baby {
    static int numBabiesMade = 0;
}
Baby.numBabiesMade = 100;
Baby b1 = new Baby();
Baby b2 = new Baby();
Baby.numBabiesMade = 2;
```

What is

`b1.numBabiesMade`?

`b2.numBabiesMade`?

static example

- Keep track of the number of babies that have been made.

```
public class Baby {
    int numBabiesMade = 0;
    Baby() {
        numBabiesMade += 1;
    }
}
```

static field

- Keep track of the number of babies that have been made.

```
public class Baby {
    static int numBabiesMade = 0;
    Baby() {
        numBabiesMade += 1;
    }
}
```

static method

```
public class Baby {
  static void cry(Baby thebaby) {
    System.out.println(thebaby.name + "cries");
  }
}
```

Or

```
public class Baby {
  void cry() {
    System.out.println(name + "cries");
  }
}
```

static notes

- Non-static methods can reference static methods, but not the other way around
  - Why?

```
public class Baby {
    String name = “DMX”;
    static void whoami() {
        System.out.println(name);
    }
}
```

main

- Why is main static?

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

Assignment 4

- Modeling Book and Libraries
  - class Book {}
  - class Library{}
- Books can be
  - Borrowed
  - Returned
- Library
  - Keeps track of books
  - Hint: use Book[]

6.092 Introduction to Programming in Java

January (IAP) 2010