204700
Data Structure and Programming Languages

Jakarin Chawachat

From: http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/index.htm

Solution 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;
}
```

Solution 2

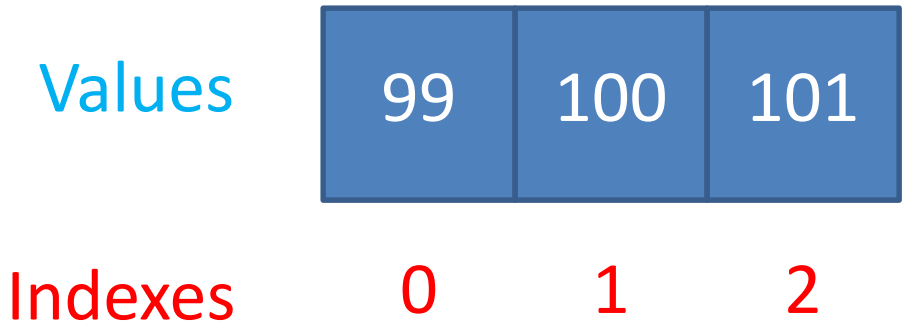
```
public static int getSecondMinIndex(int[] values) {
        int secondIdx = -1;
        int minIdx= getMinIndex(values);
        for(int i=0; i<values.length; i++) {
                if (i == minIdx)
                        continue;
                if (secondIdx == -1 || values[i] < values[secondIdx])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



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]
                }
        }
}
```

Problem?

What if vals = {1,2,3}?

Set min = Integer.MAX_VALUE or vals[0]

Popular Issues 4

Variable Initialization – secondMinIndex

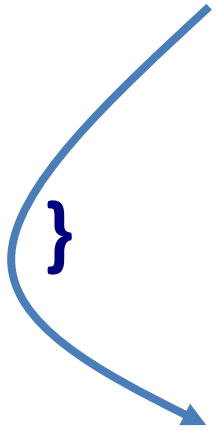
```
int minIdx = getMin(vals)
        int secondIdx = 0;
        for (int i =0; i <vals.length; i++) {
                if (i == minIdx) continue;
                if (vals[i] < vals[secondIdx])
                secondIdx = i;
         }
}
```

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

Popular Issues 5

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

```
public class fibonacci {
   public static void fibo(int n){
      int a0 = 0;
      int a1 = 1;
      int count = 2;
      System.out.print(a0 + " " + a1);
      while(count < n){
         int ans = a0 + a1;
         System.out.print(" " + ans);
         a0 = a1;
         a1 = ans;
         count++;
     }
      System.out.println("");
  }
   public static void main(String[] args) {
      fibo(5);
      fibo(15);
      fibo(10);
  }
}
```

Outline

- Object oriented programming
- Defining Classes
- Using Classes
- References vs Values
- Static types and methods

Object oriented programming

- Represent the real world

Baby

Name

Sex

Weight

….

Object oriented programming

Objects group together

– Primitives (int, double, char, etc..)

– Objects (String, etc…)

Baby

String Name

boolean isMale

double weight

….

Why use classes?

Why not just primitives? 2 babies: Alex and David

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

Why use classes?

Why not just primitives? 2 babies: Alex and David

```
// little baby alex
String nameAlex;
double weightAlex;
// little baby david
String nameDavid;
double weightDavid;
// little baby david2
String nameDavid2;
double weightDavid2;
```

500 Babies?

David2?

Why use classes?

Baby 1

Why use classes?

497 more Babies

DEFINING CLASSES

Class -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

Class - overview

Baby myBaby **= new** Baby**();**

Class
Instance

Declare a baby

```
public class Baby {
}
```

Note

- Class name 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;
        boobean isMale;
        int numPoops = 0;

}
```

Baby Siblings?

```
public class Baby {
        String name;
        double weight = 5.0;
        boolean isMale;
        int numPoops =0;
        xxxxx yyyy;
        Baby[] sibling;
}
```

Let’s make this baby!

Baby ourBaby **= new** Baby**();**

But what about it’s name? it’s 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 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 {
        double weigth = 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;
        int numPoops = 0;
        Baby[] siblings;

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

USING CLASS

Classes and Instances

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


//class Instances
Baby steven = new Baby("Steven Joel", true);
Baby king = new Baby("King Joel", true);
```

Accessing fields

- Object.FIELDNAME

```
Baby steven = new Baby("Steven Joel", true);

System.out.println(steven.name);
System.out.println(steven.numPoops);
```

Calling Methods

- Objext.METHODNAME([ARGUMENTS])

```
Baby steven = new Baby("Steven Joel", true);

steven.sayHi();   // “Hi, my name is Steven Joel”
steven.eat(1);
```

REFERENCE AND VALUES

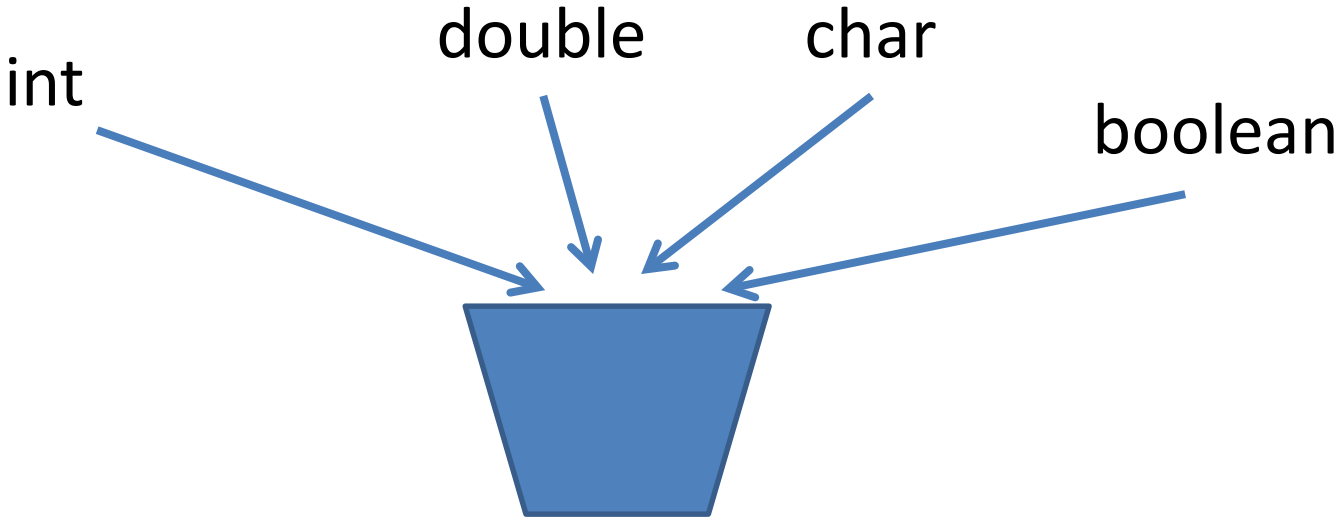
Primitives and References

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

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

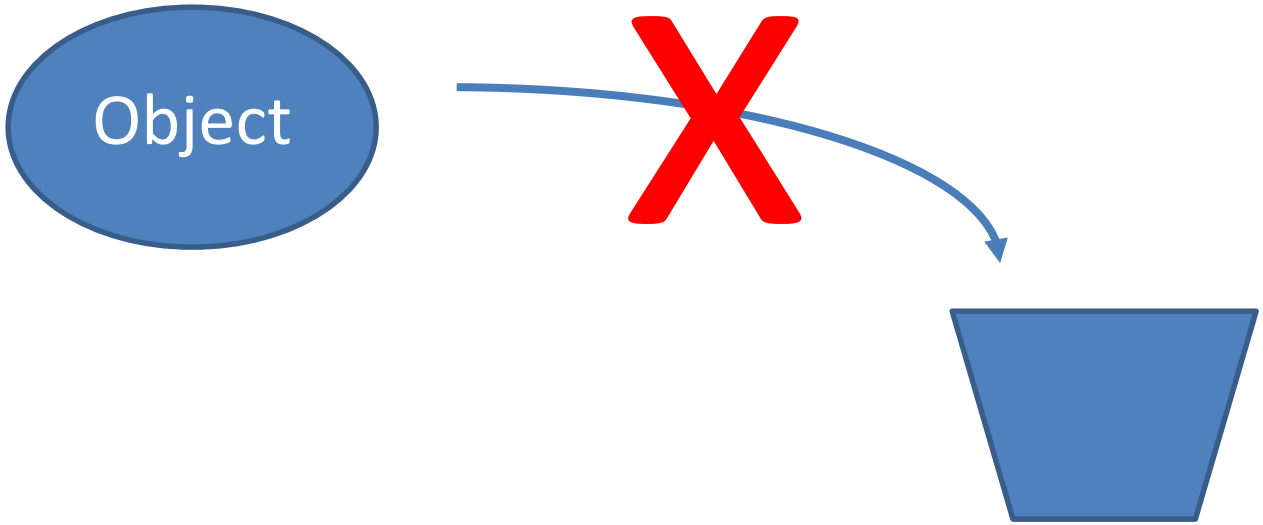
How java stores primitives

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



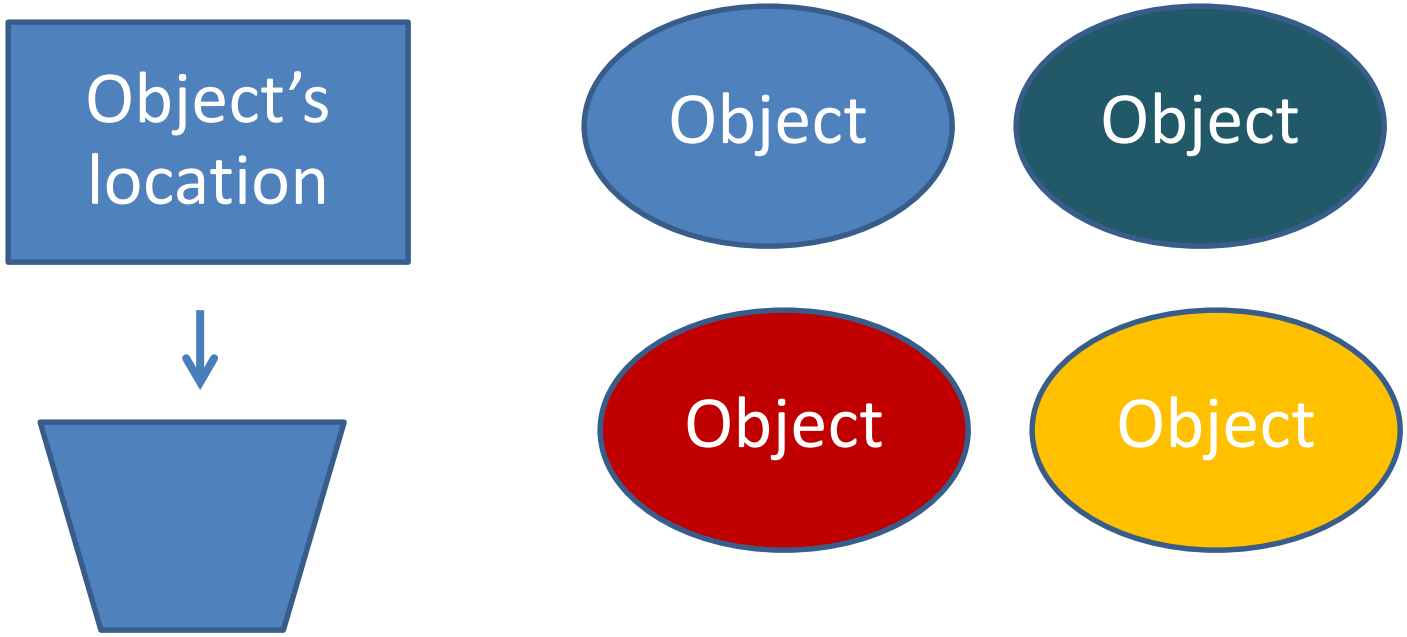
How java store objects

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



How java store 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 reference

== compares the references

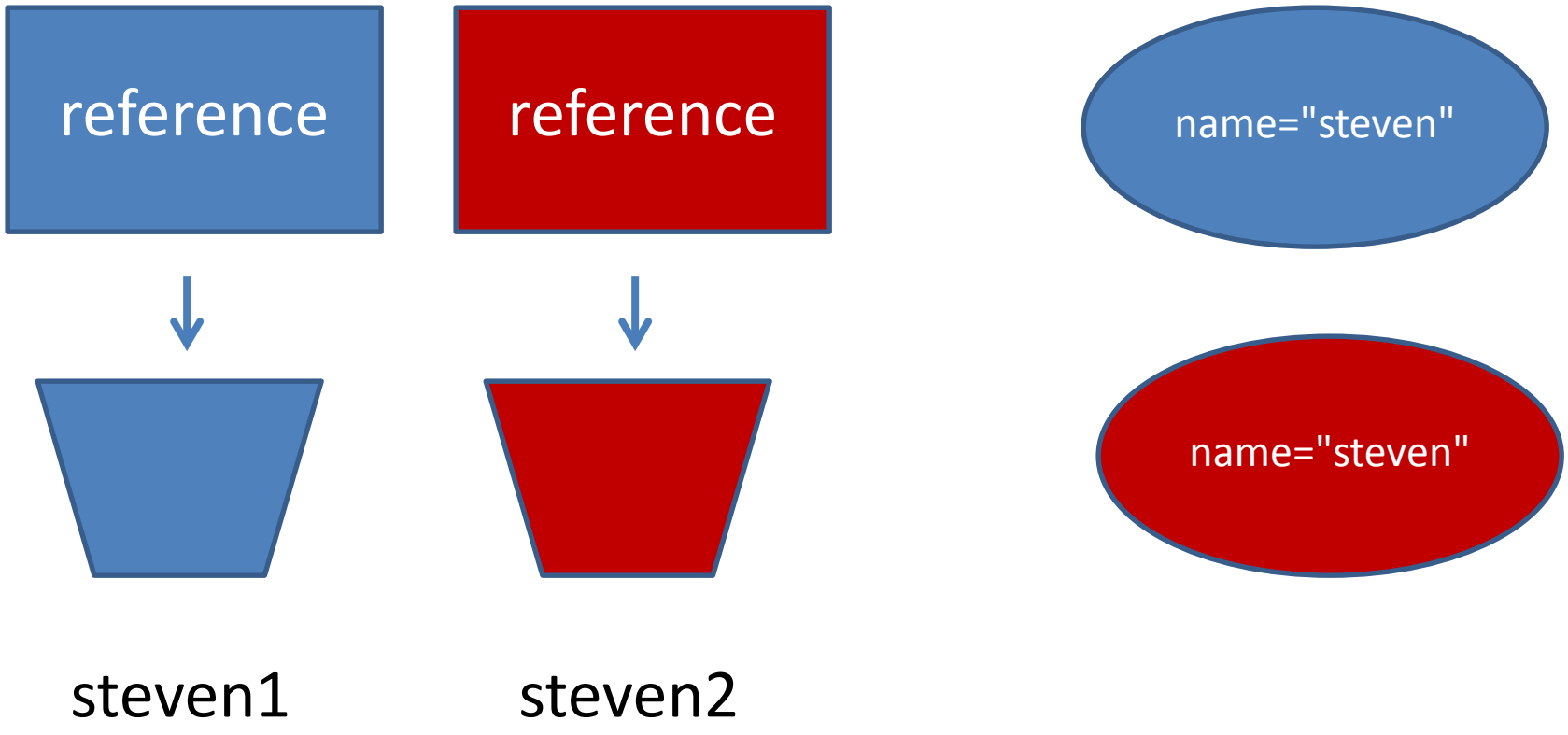
```
Baby steven1 = new Baby("steven");
Baby steven2 = new Baby("steven");
```

Does steven1 == steven2?

No

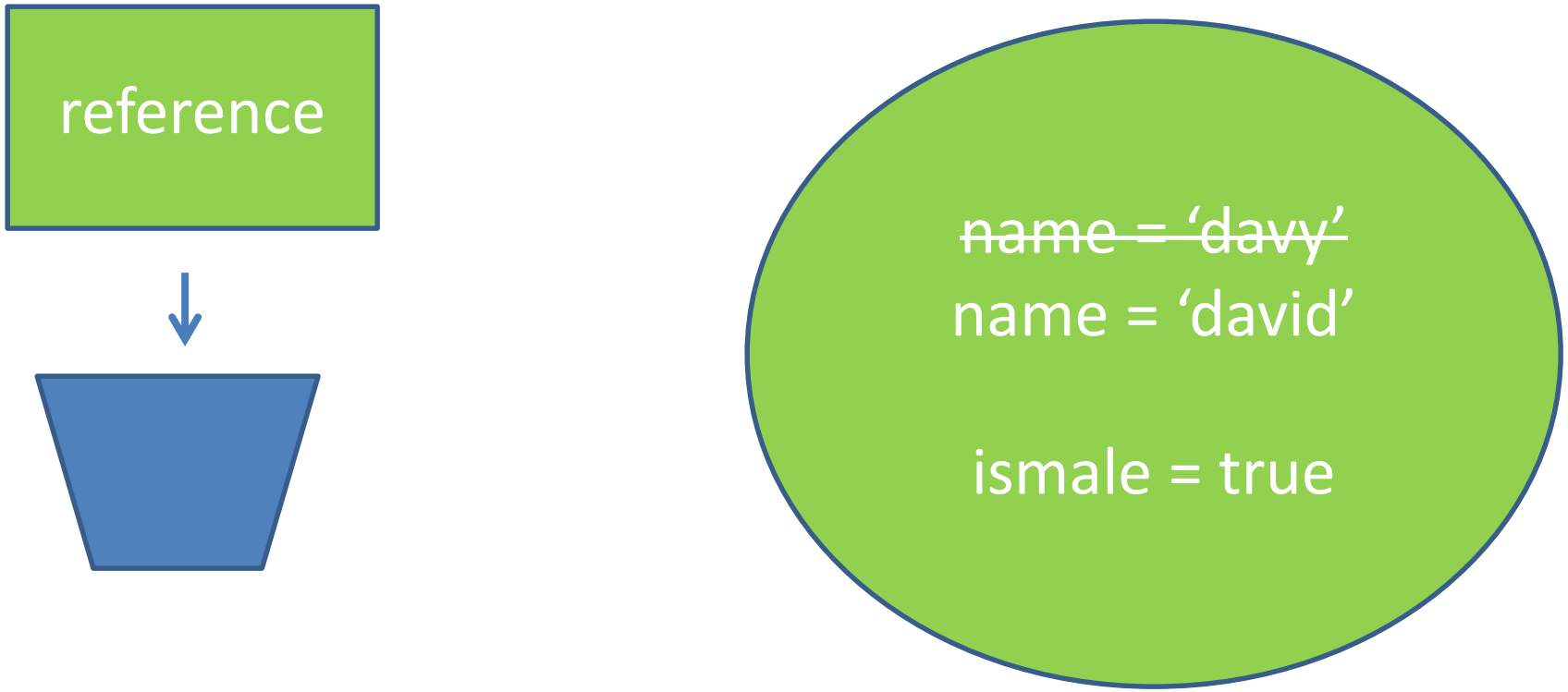
References

```
Baby steven1 = new Baby("steven");
Baby steven2 = new Baby("steven");
```



References

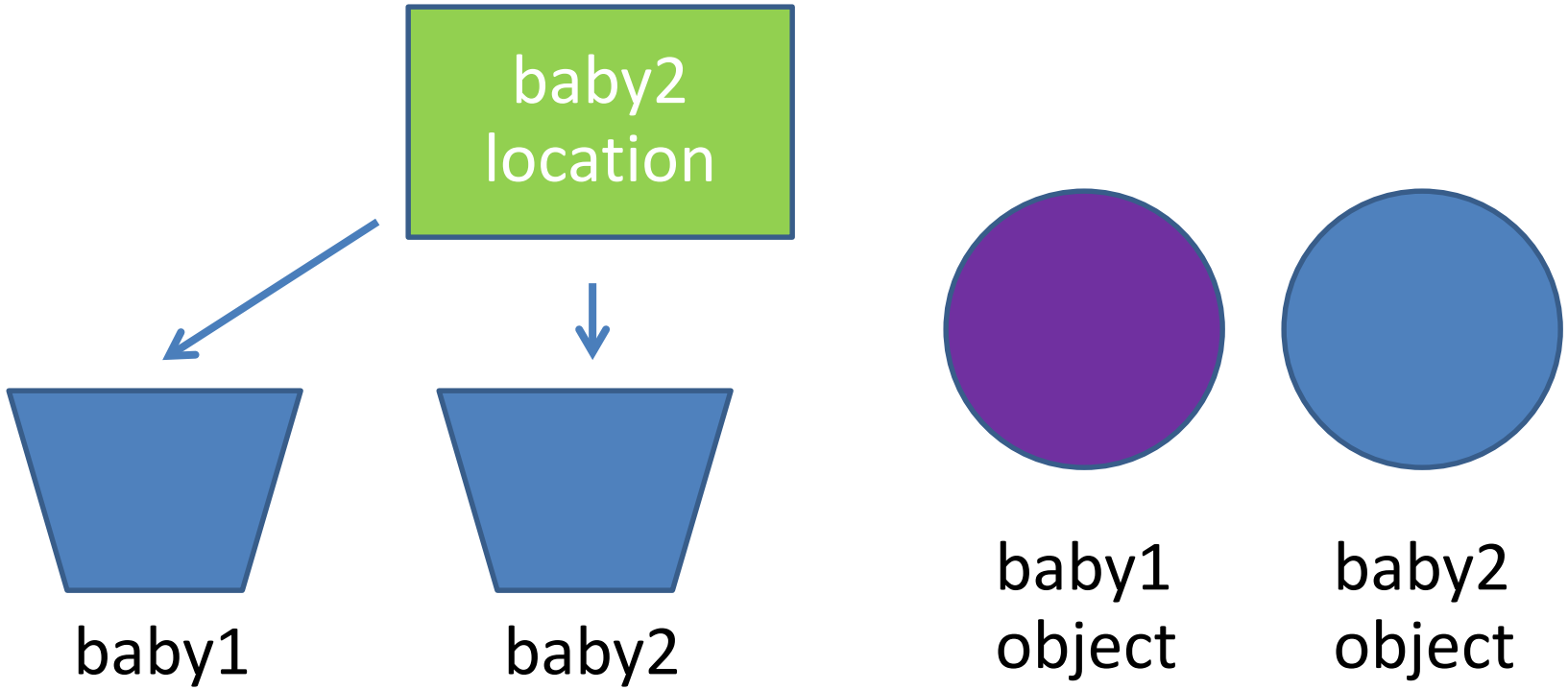
```
Baby mybaby = new Baby("davy", true);
mybaby.name = "david";
```



References

Using = updates the reference.

baby1 = baby2



STATIC TYPES AND METHODS

static

- Applies to field 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);
        }
}
```

static notes

- A static method belongs to the class itself and a non-static (aka instance) method belongs to each object that is generated from that class.
- If your method does something that doesn't depend on the individual characteristics of its class, make it static (it will make the program's footprint smaller). Otherwise, it should be non-static.

```
class Foo {
   int i;

   public Foo(int i) {
     this.i = i;
   }

   public static String method1() {
     return "An example string that doesn't depend on i (an instance
variable)";
   }

   public int method2() {
     return this.i+1; // Depends on i
   }
}
```

You can call static methods like this: Foo.method1(). If you try that with method2, it will fail. But this will work: Foo bar = new Foo(1); bar.method2();

Exercise

- create circle class
- test call in main

Assignment 5

- Create Triangle class
  - double height
  - double base
  - method findArea()
- In main method
  - Triangle t1 = new Triangle();
  - Triangle t2 = new Triangle(5,10);
  - T1.findArea();
  - T2.findArea();