Visual Programming

Lecture 4: Classes and Objects

Mahmoud El-Gayyar

elgayyar@ci.suez.edu.eg

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 minIdx) {
        int secondIdx = -1;


        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

- 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 2

- 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]`

Outline

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

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

Baby

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

Why use classes?

- Why not just primitives?

```
// little baby ali
String nameAli;
double weightAli;
// little baby mona
String nameMona;
double weightMona;
```

Primitives!!

- Why not just primitives?

```
// little baby ali
String nameAli;
double weightAli;

// little baby mona
String nameMona;
double weightMona;

// little baby mona
String nameMona2;
double weightMona2;
```

Terrible 

500 Babies?

Why use classes?

Baby1

Why use classes?

Baby1

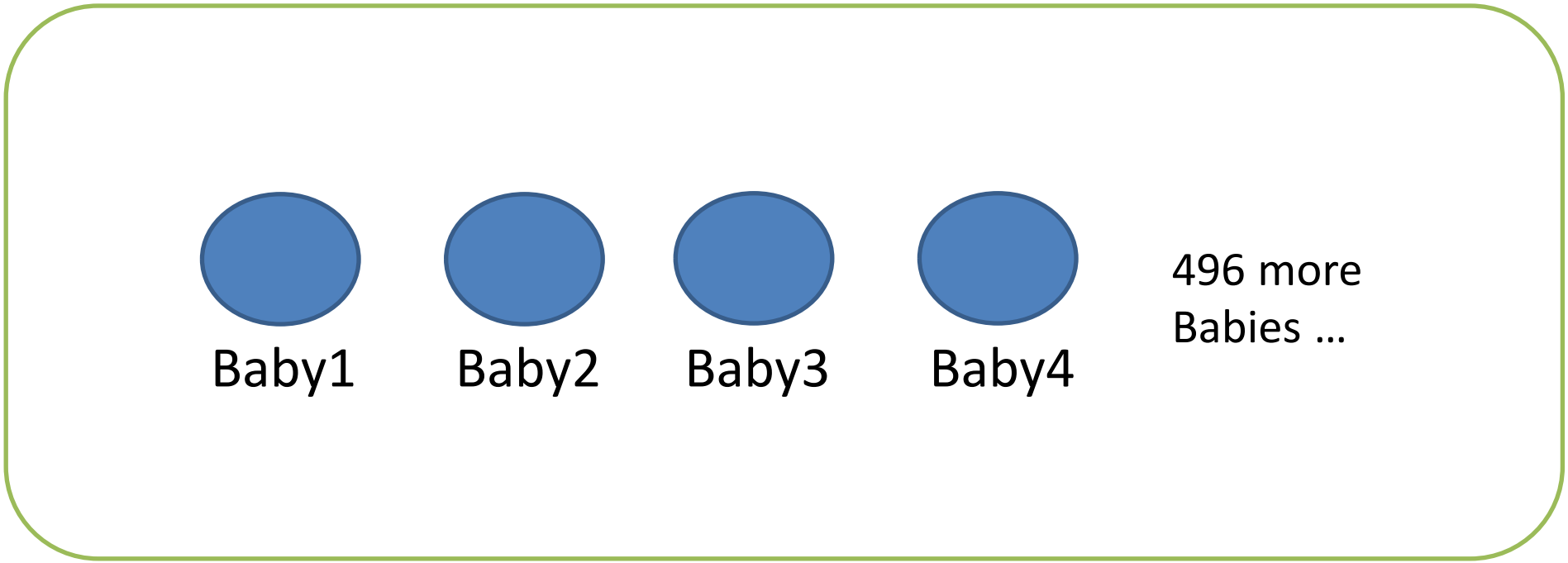
Baby2

Baby3

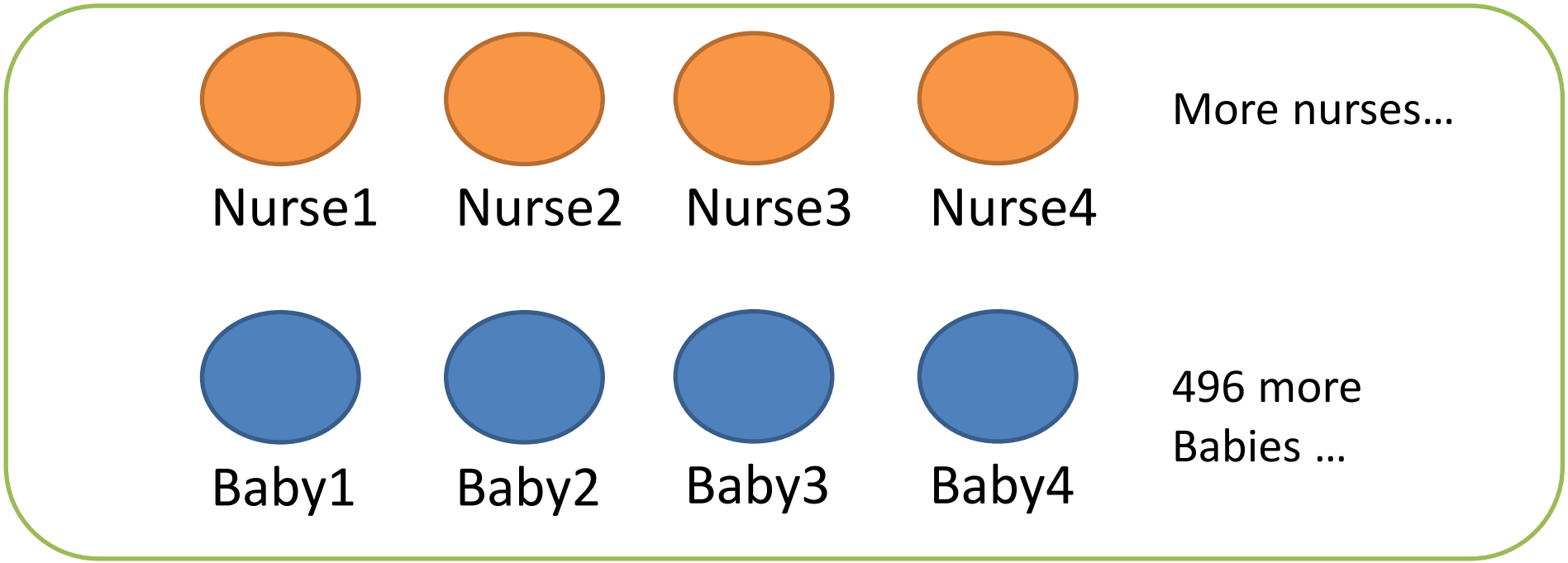
Baby4

496 more Babies …

Why use classes?

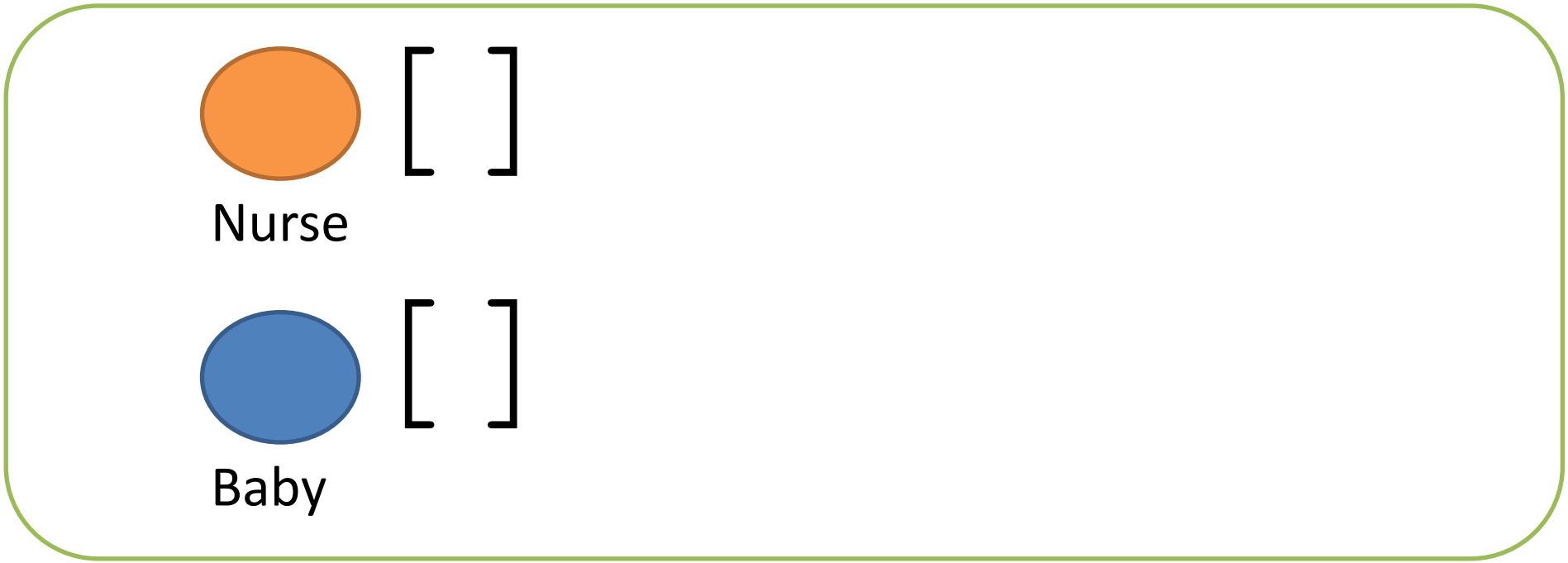


Why use classes?



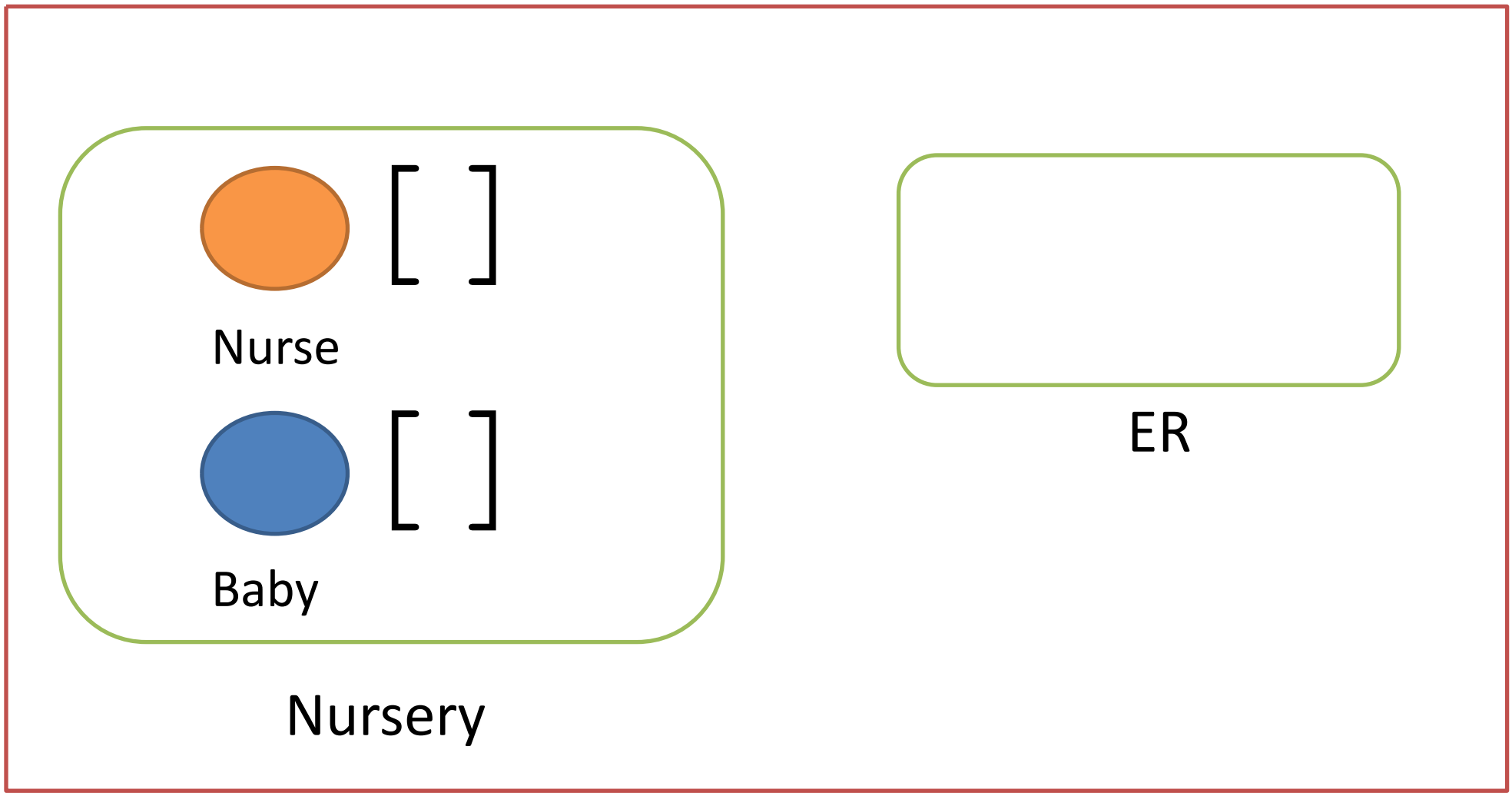
Nursery

Why use classes?



Nursery

Why use classes?



Outline

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

Class - Overview

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


    void sayHi() {

      System.out.println("Hi");


    }
}
```

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 weight?

Constructors

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

    public     CLASSNAME  ([ARGUMENTS])  {
    }
}
```

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

Constructors

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

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

Baby Constructor

```
public class Baby {
    String name;
    boolean isMale;
     public  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) {…}
}
```

Outline

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

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(knox.name);
```

Calling Methods

- Object.METHODNAME([ARGUMENTS])

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

Outline

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

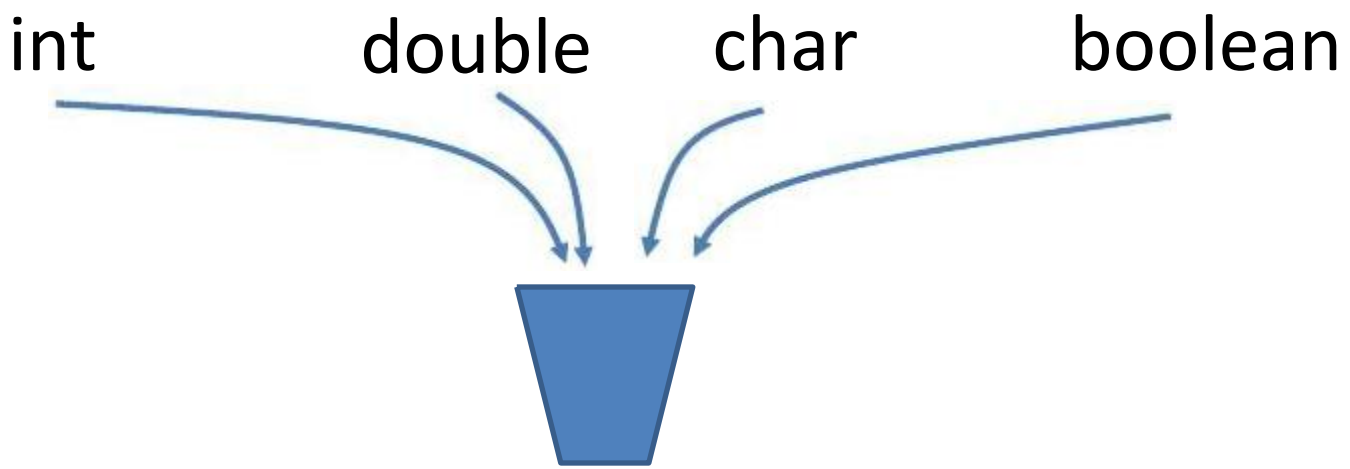
Primitives vs. References

- 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, …

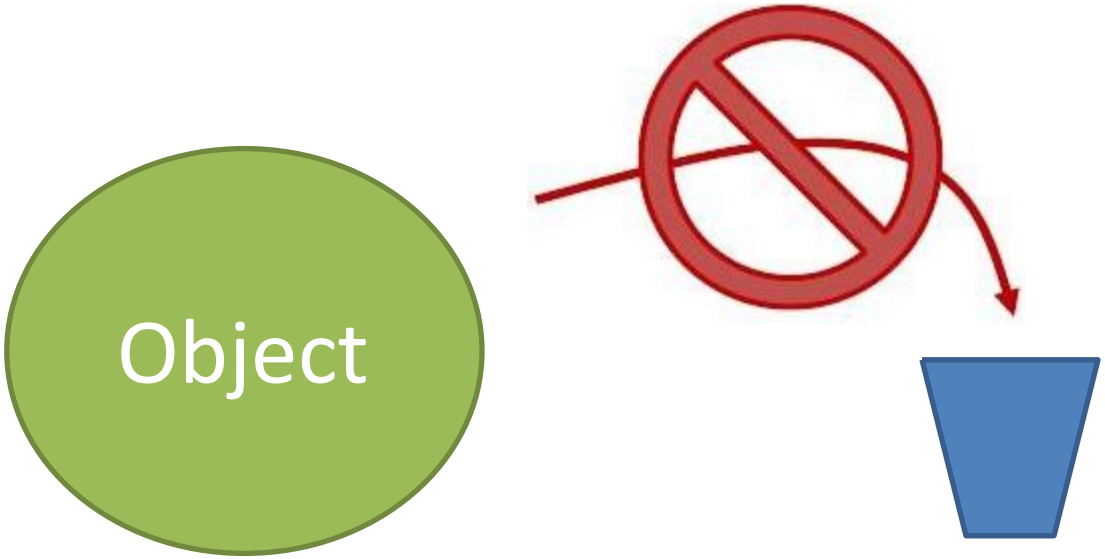
How Java stores Primitives

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



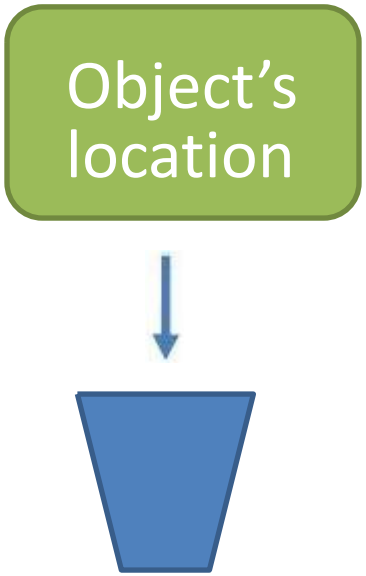
How Java stores Objects

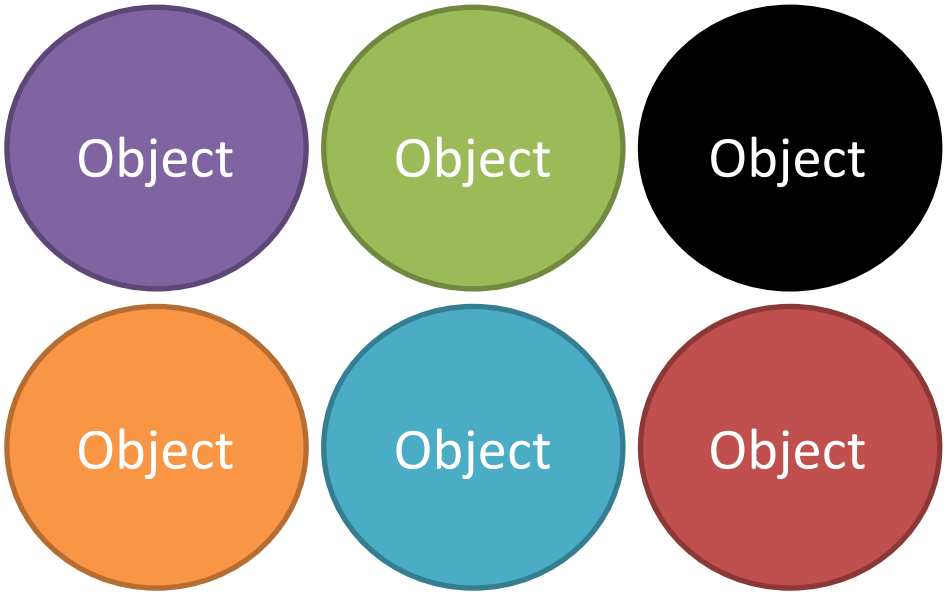
- Objects are too big to fit in a variable
  - Stored somewhere else
  - Variable stores a number that locates the object



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

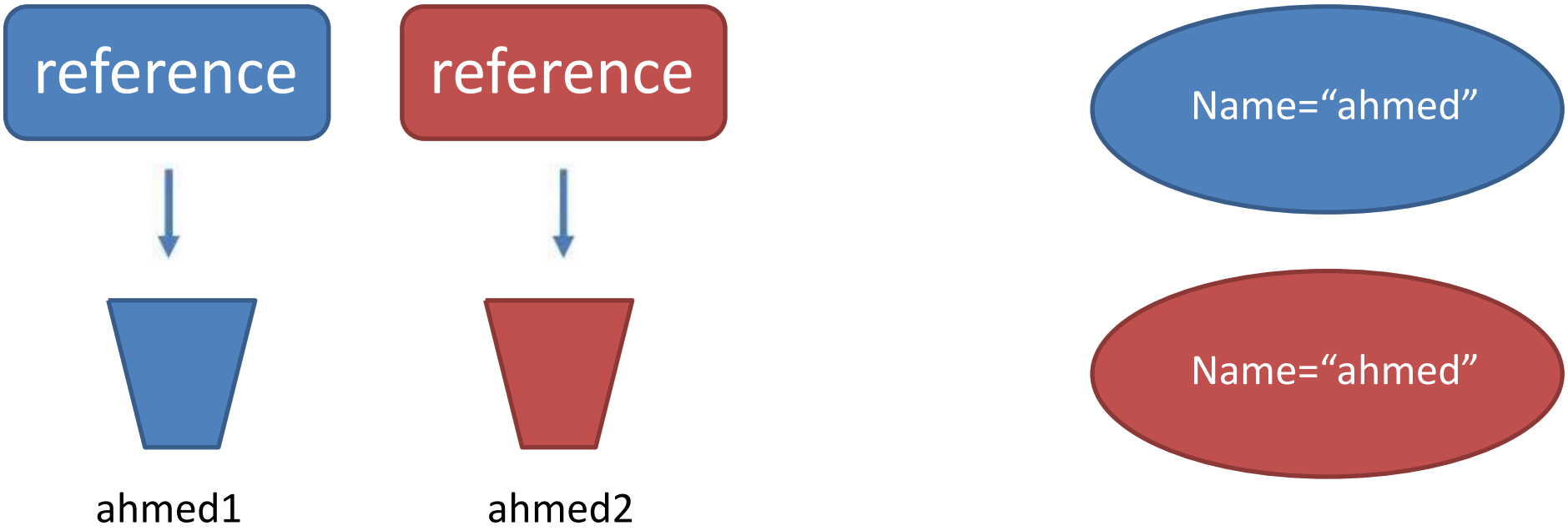
```
Baby ahmed1 = new  Baby("ahmed");
Baby ahmed2= new  Baby("ahmed");
```

Does ahmed1==ahmed2

no

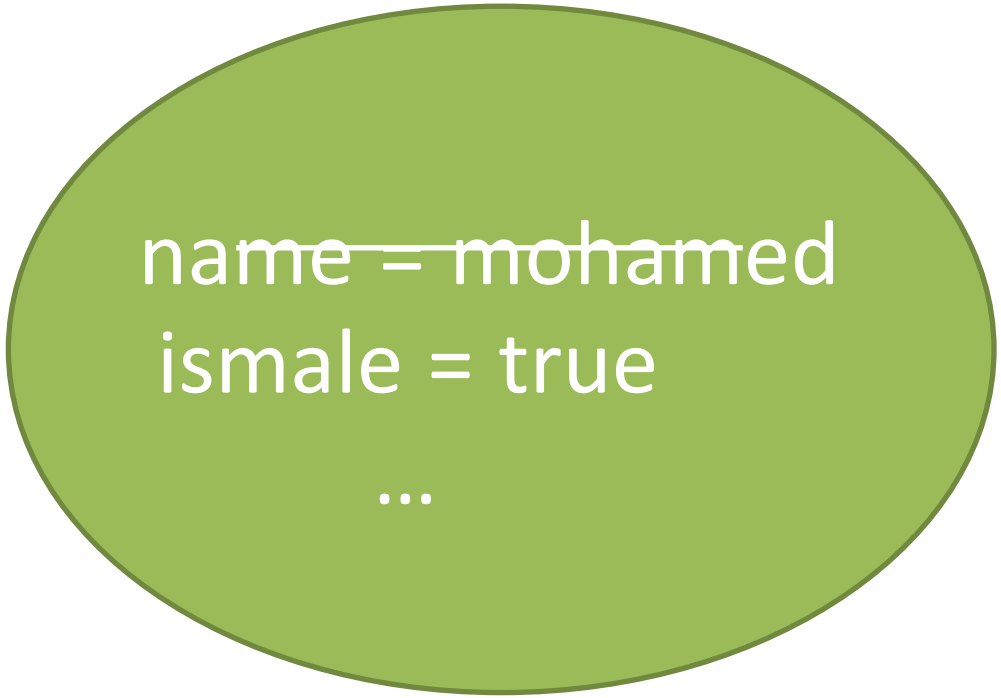
References

```
Baby ahmed1= new Baby("ahmed");
Baby ahmed2= new  Baby("ahmed");
```



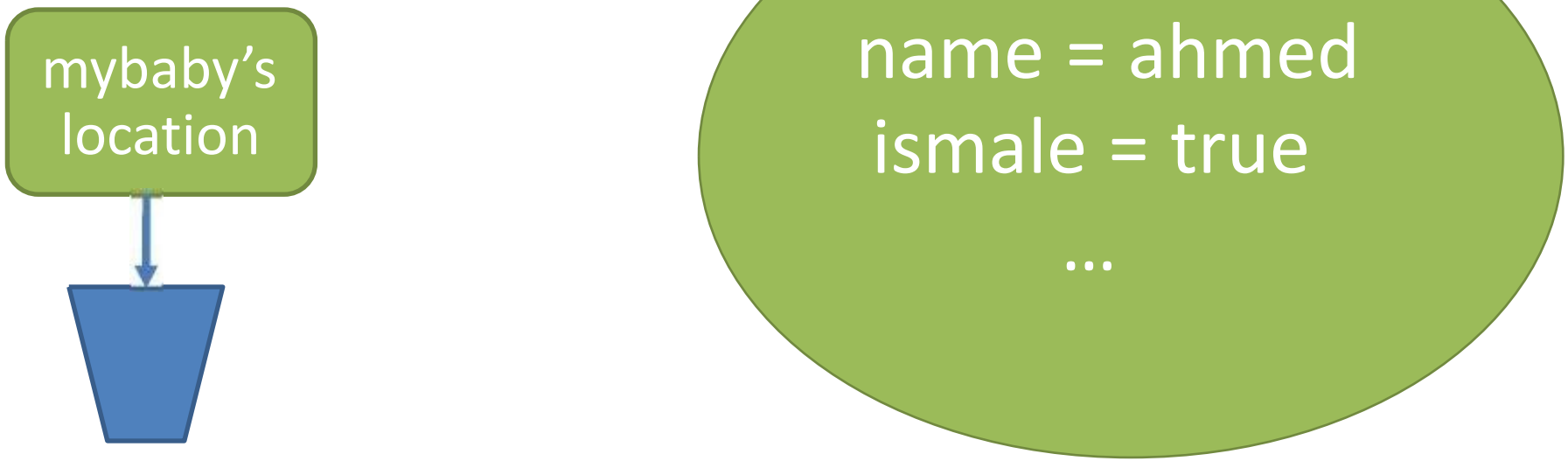
References

```
Baby mybaby = new Baby("mohamed", true)
mybaby.name = "ahmed"
```



References

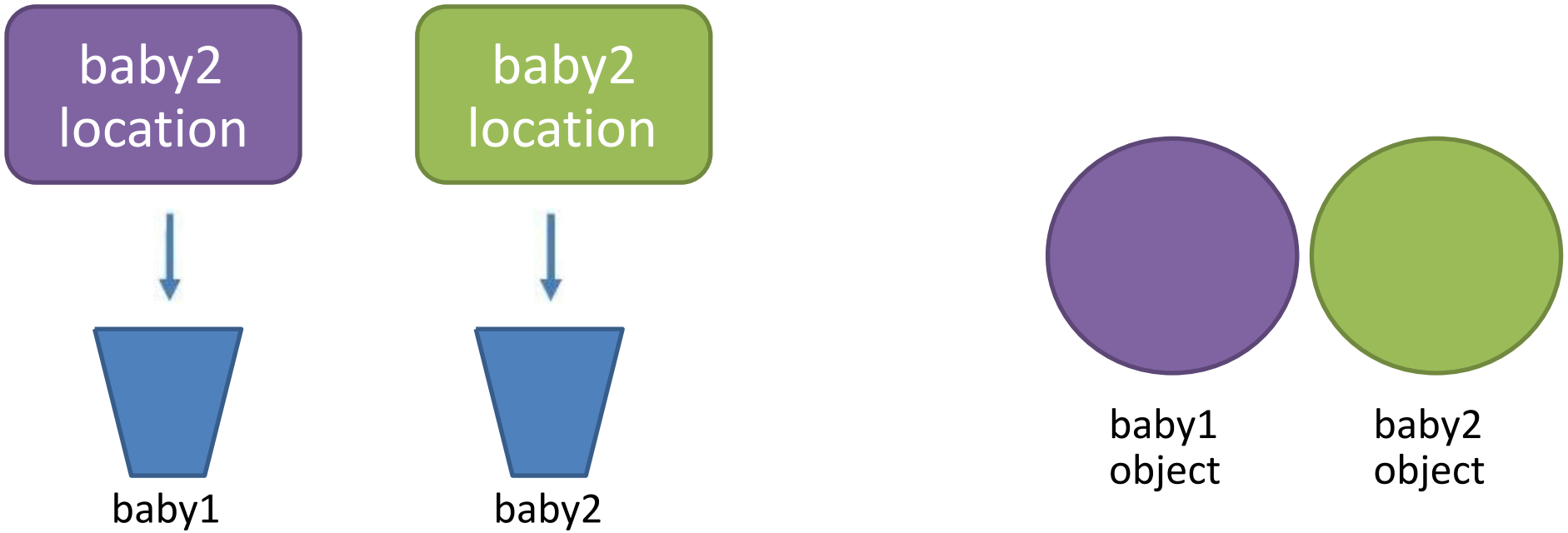
```
Baby mybaby = new Baby("mohamed", true)
mybaby.name = "ahmed"
```



References

- Using = updates the reference.

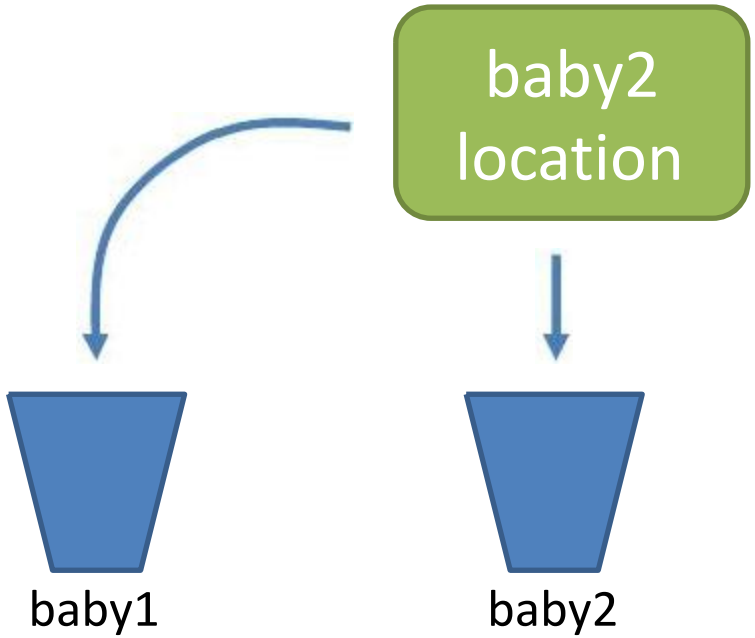
```
baby1 = baby2
```

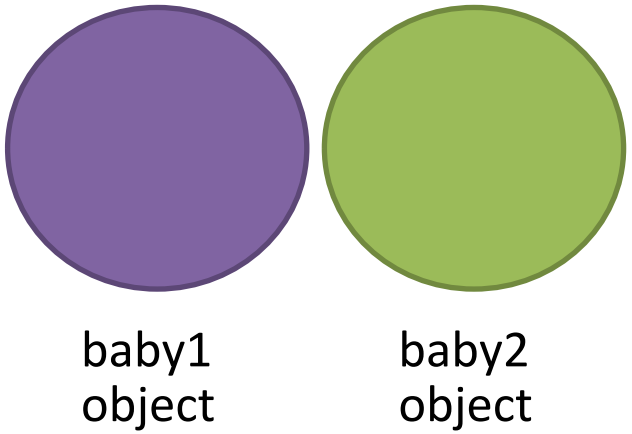


References

- Using = updates the reference.

```
baby1 = baby2
```





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=?

Outline

- *Object oriented programming*
- *Defining Classes*
- *Using Classes*
- *References vs. Values*
- ***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 numBabies = 0;
}
Baby.numBabies = 100;
Baby b1 = new Baby();
Baby b2 = new Baby();
Baby.numBabies = 2;
```

What is

b1.numBabies?

b2.numBabies?

Static Example

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

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

Static Field

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

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

Static Method

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

OR

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

A static method is a class method and is not attached to an object. So if we use a non static variable of the class, it would most probably not have been initialized because no object could have been created for the class. Hence it would throw a null pointer exception.

Main Method

- Why is main static?

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

- The main() method in Java is static because they can then be invoked by the runtime engine ***without having to instantiate an instance*** of the parent class. (***which Constructor***)

Summary

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