

# Programming 1

## Lecture 6 – OOP Classes & Objects

# Object-Oriented Programming

- Model real-life objects in a meaningful way.
  - Objects which possess properties and behaviors
  - E.g. A certain dog has yellow fur, blue eyes... it can run, bark and wiggle its tail...
- A clear structure for the program.
  - Easier to maintain, modify and debug
  - “Don't Repeat Yourself” principle

# Class vs. Instance

**Class**

**Puppy**

**Apple**



**Instances**



# Java Objects & Classes

- Object is a more general term than instance
  - > In Java, object = instance
- You have encountered different objects
  - `System.out` -- a `PrintStream` object
  - `"Hello, world"` -- a `String` object
  - `Scanner sc` -- a `Scanner` object
- Each **object** belongs to a certain **class**
- An object has attributes and methods
  - **Defined in the class** to which it belongs

# Why use objects?

- Why not just primitives?

```
// little baby alex
```

```
String nameAlex;
```

```
double weightAlex;
```

```
// little baby david
```

```
String nameDavid;
```

```
double weightDavid;
```

# Why use objects?

```
// little baby alex
```

```
String nameAlex;
```

```
double weightAlex;
```

```
// little baby david
```

```
String nameDavid;
```

```
double weightDavid;
```

```
// little baby david
```

```
String nameDavid2;
```

```
double weightDavid2;
```

**Many babies?**

Hard to manage

# Why use objects?



Baby1

## Managing many babies?

We can use arrays for *name*, *weight* & *sex*

...but objects are just better



Baby1



Baby2



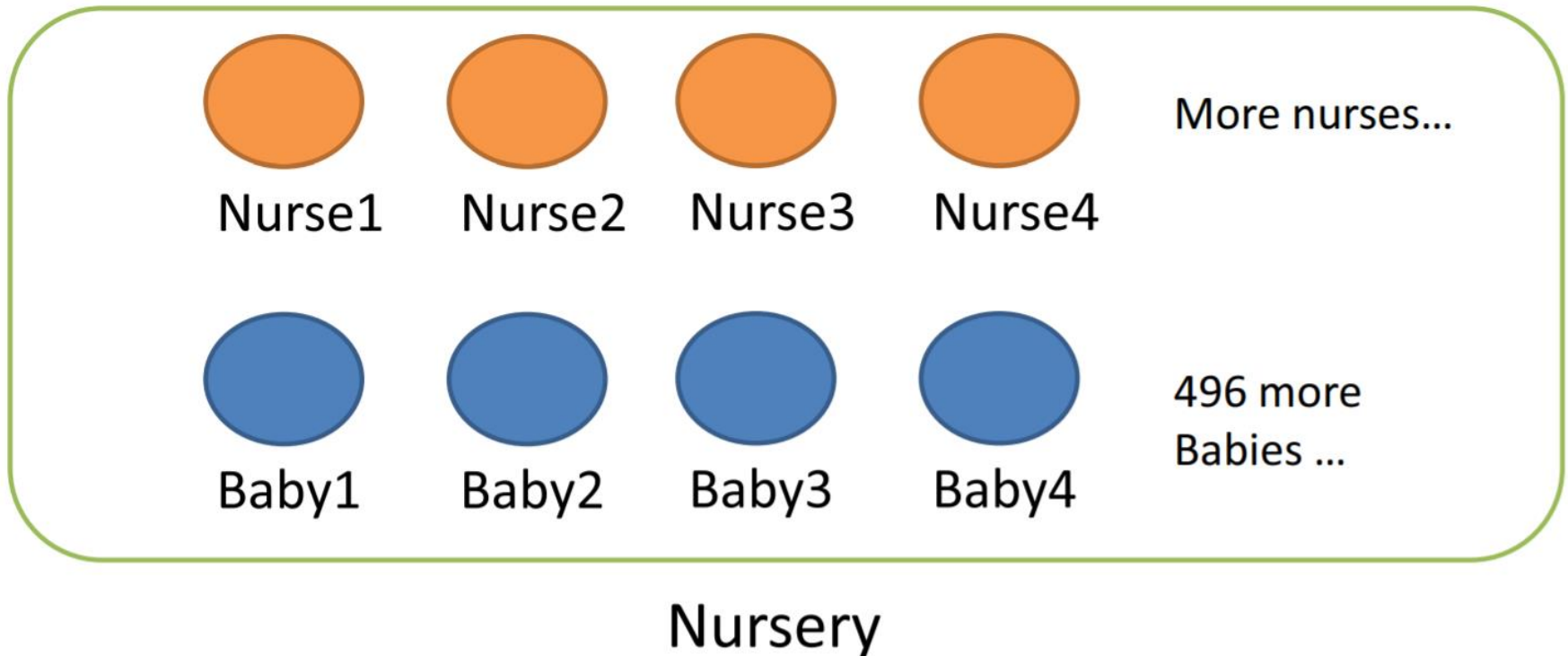
Baby3



Baby4

# Why use objects?

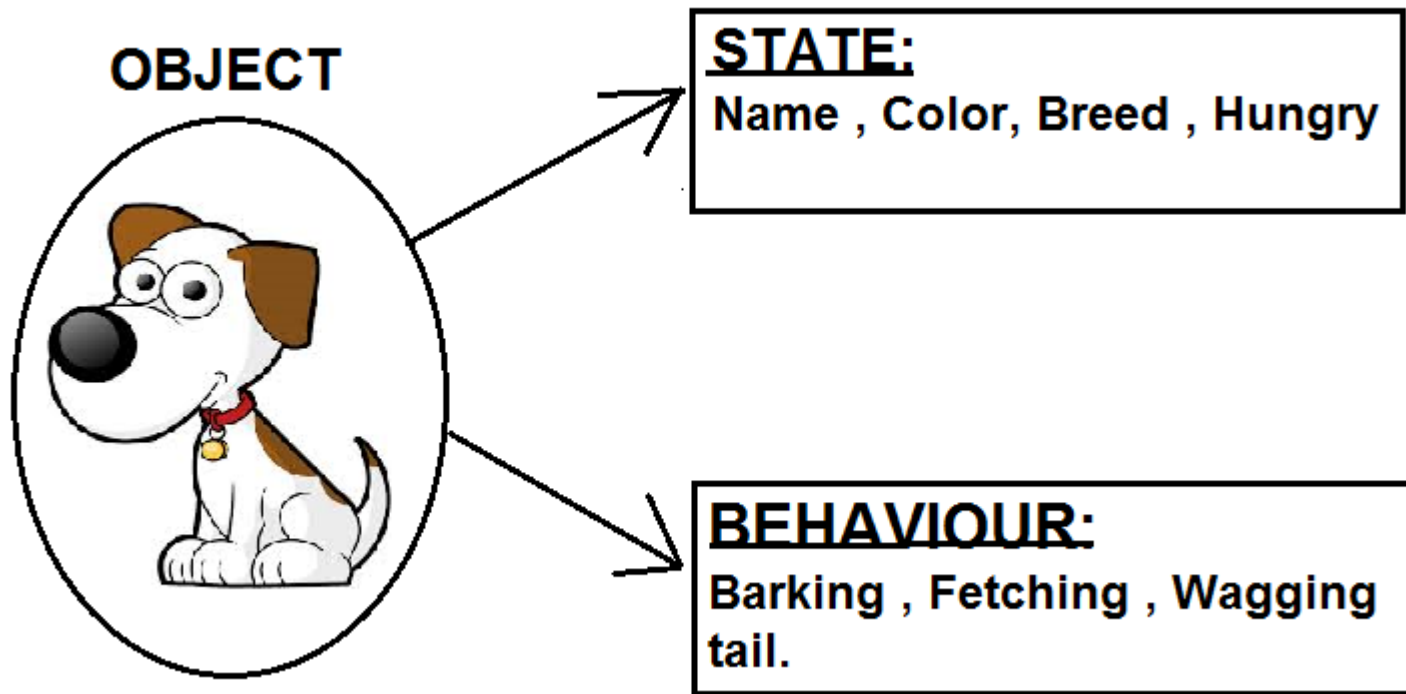
Even better when we need to model relationships





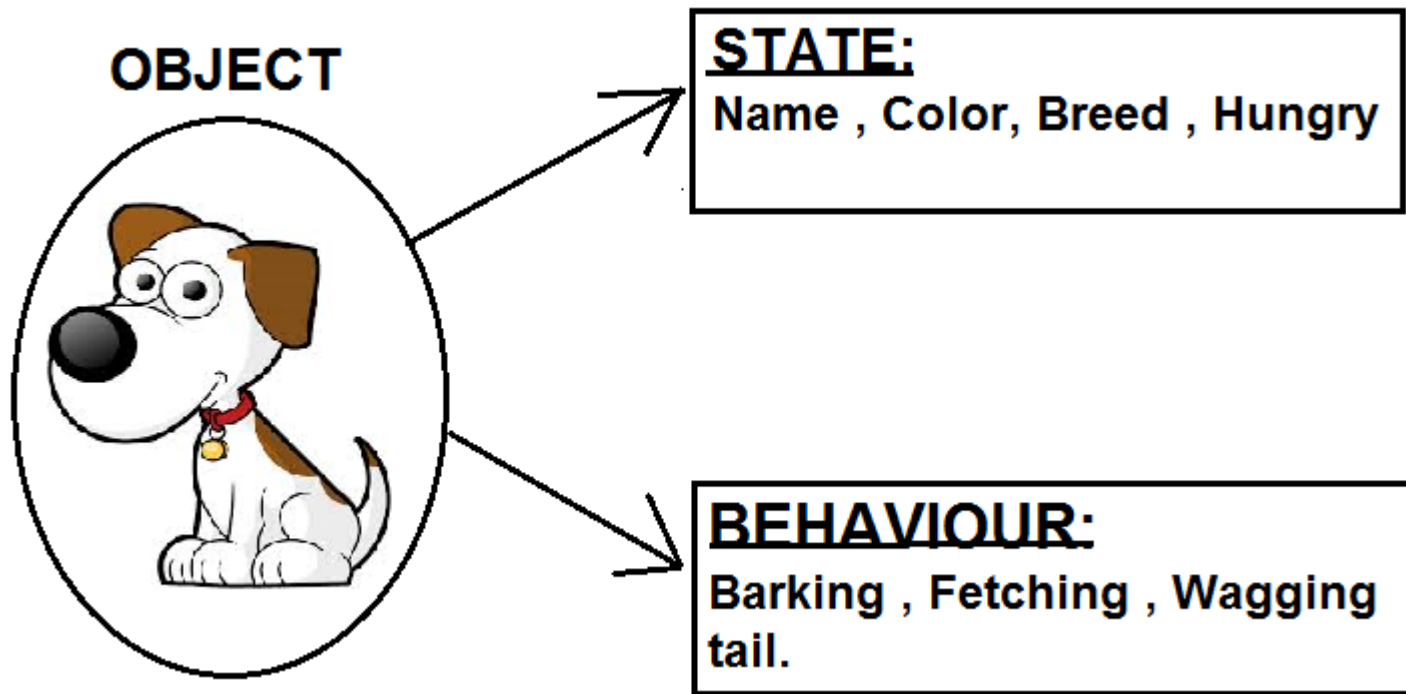
# Why use objects?

- Objects can also have behaviors



# Why use objects?

- Objects can also have behaviors



# Why use objects?

- A program contains many objects which have different functions and interact with each other.
- Similarly, home appliances have different functions and all contribute to the operation of a household.



© Luc Meaille/iStockphoto.

# Class -- Overview

```
public class Baby {  
    String name;  
    boolean isMale;  
    double weight;  
    double decibels;  
    int numPoops = 0;  
  
    public void poop() {  
        numPoops += 1;  
        System.out.println("Dear mother, " +  
            "I have pooped. Ready the diaper.");  
    }  
}
```

Class Definition

# Class -- Overview

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

Creating class **Instance**

# Let's declare a class

```
public class Baby {
```



fields



methods

```
}
```

# Notes

- The *public* class `Baby` must be defined in the file `Baby.java`
- The *main* method is not part of the class definition.
  - It belongs to the *static* context.
  - The *main* method inside a class can be run.
  - Although being in the same class, the *main* method is not related to the non-static methods and attributes in any way.

# Baby fields

- **Fields** or **attributes** are used to store data.

```
public class Baby {  
    String name;  
    double weight = 5.0;  
    boolean isMale;  
    int numPoops = 0;  
}
```



# Baby siblings?

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

- A class **attribute** can be any type
  - Primitives, class instances (including instances of the class itself)
  - Array of primitives, array of objects

# How to store/modify data in object?

- Direct access

- Requires fields to be *public* or *default*.
- Does not work with *private* or *protected* fields.

```
Baby myBaby = new Baby();  
myBaby.name = "Alex";  
myBaby.isMale = true;
```

- Use a constructor

- Need to define a suitable constructor in the class

```
// create a male baby named Alex  
Baby myBaby = new Baby("Alex", true);
```

# Constructors

```
public class Baby {  
    public Baby () {  
        // nothing  
    }  
    public Baby(String name, boolean isMale) {  
        // some codes  
    }  
}
```

```
Baby b1 = new Baby();  
Baby b2 = new Baby("Alex", true);
```

A constructor is invoked when an instance is created.

# Constructors

- A constructor is like a method
- Constructor name is the same as class name
  - No return type – never returns anything
- Usually initialize fields
- All classes need at least one constructor
  - There can be more than one constructor.
  - There is a default constructor even if you don't write one.

```
public ClassName () {  
    // default constructor  
}
```

# Baby constructor

```
public class Baby {  
    String name;  
    boolean isMale;  
    public Baby(String n, boolean m) {  
        name = n;  
        isMale = m;  
    }  
}
```

# Baby methods

```
public class Baby {  
    String name;  
    public Baby(..) {...}  
  
    public void sayHi() {  
        System.out.println("Hi, my name is " + name);  
    }  
  
    public void eat(double food) {  
        if (food >= 3) {  
            System.out.println(name + " is full");  
        } else {  
            System.out.println(name + " is still hungry");  
        }  
    }  
}
```

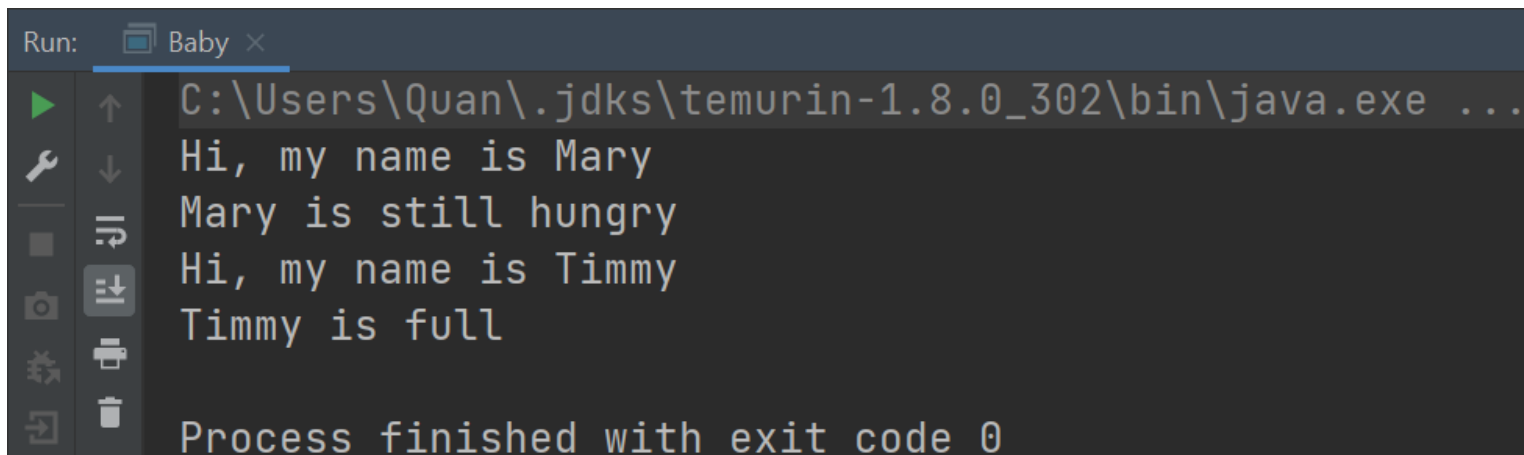
# Baby class

```
public class Baby {  
    // fields  
    String name;  
    double weight;  
    boolean isMale;  
    int numPoops;  
    Baby[] siblings;  
  
    // constructor(s)  
    public Baby(...) {...}  
  
    // methods  
    public void sayHi() {...}  
    public void eat(double food) {...}  
}
```

# Using Baby class

```
Baby mary = new Baby("Mary", false);  
mary.sayHi();  
mary.eat(1);
```

```
Baby tim = new Baby("Timmy", true);  
tim.sayHi();  
tim.eat(5);
```



The screenshot shows a Java IDE console window with a dark theme. The title bar at the top says "Run: Baby x". The console output displays the results of the code execution: "Hi, my name is Mary", "Mary is still hungry", "Hi, my name is Timmy", and "Timmy is full". At the bottom, it states "Process finished with exit code 0". On the left side of the console, there is a vertical toolbar with icons for running, stepping through, and other debugging actions.

```
Run: Baby x  
C:\Users\Quan\.jdk\temurin-1.8.0_302\bin\java.exe ...  
Hi, my name is Mary  
Mary is still hungry  
Hi, my name is Timmy  
Timmy is full  
Process finished with exit code 0
```

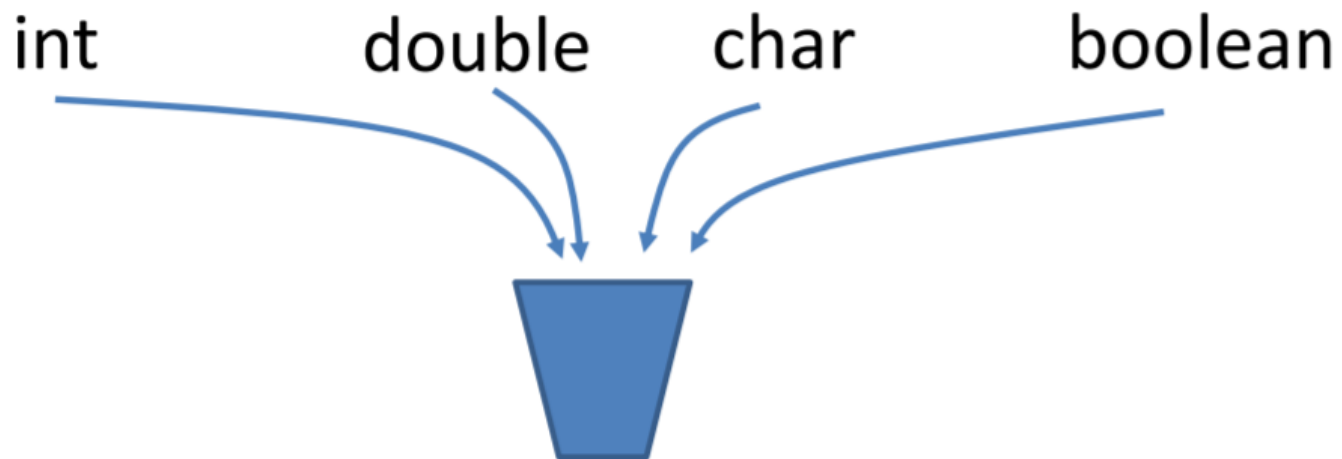


# Primitives vs References

- 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`, ...
  - The variable only stores object's memory address

# How Java stores primitives

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



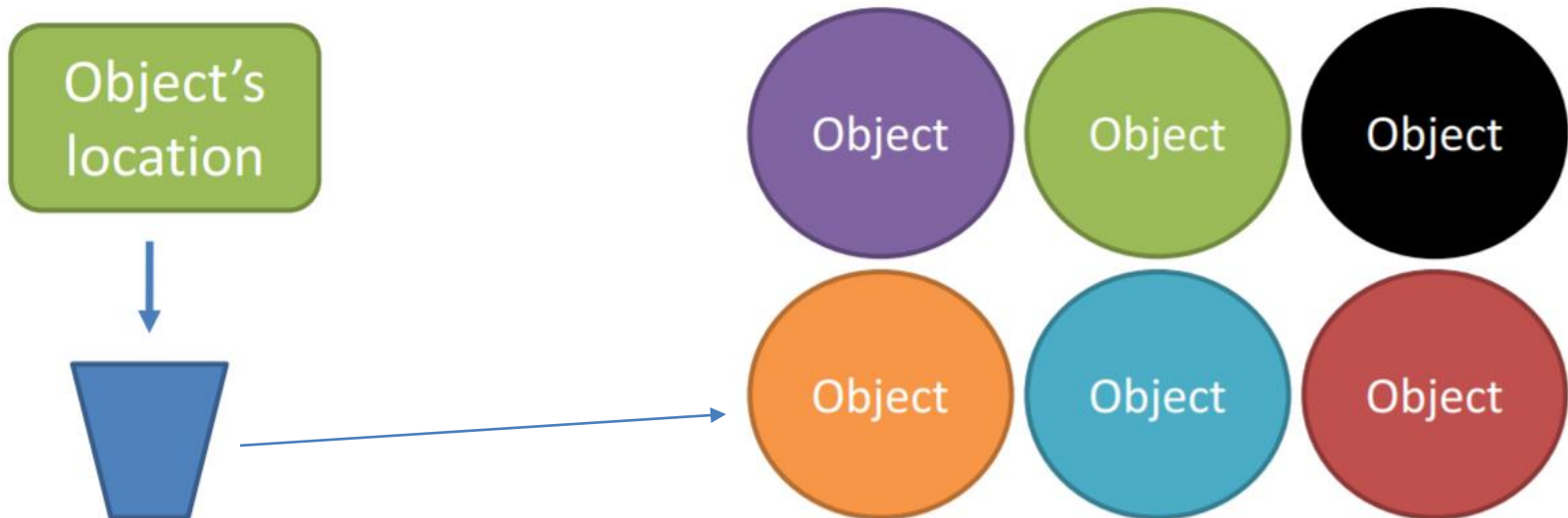
# How Java stores objects

- 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



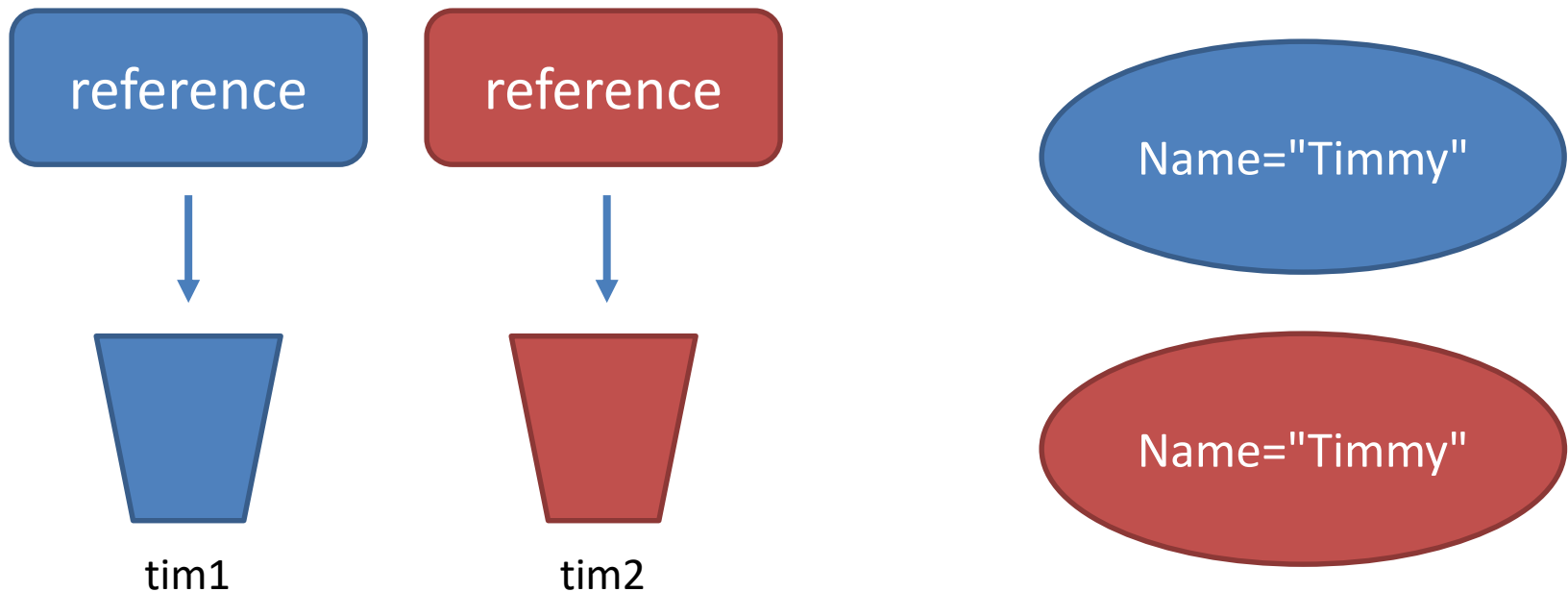
# References

- The object's location is called a reference
  - All object variables are references, as opposed to primitive variables
- `==` only compares the references

```
Baby tim1 = new Baby("Timmy");  
Baby tim2 = new Baby("Timmy");  
// true or false?  
boolean b = (tim1 == tim2);
```

# References

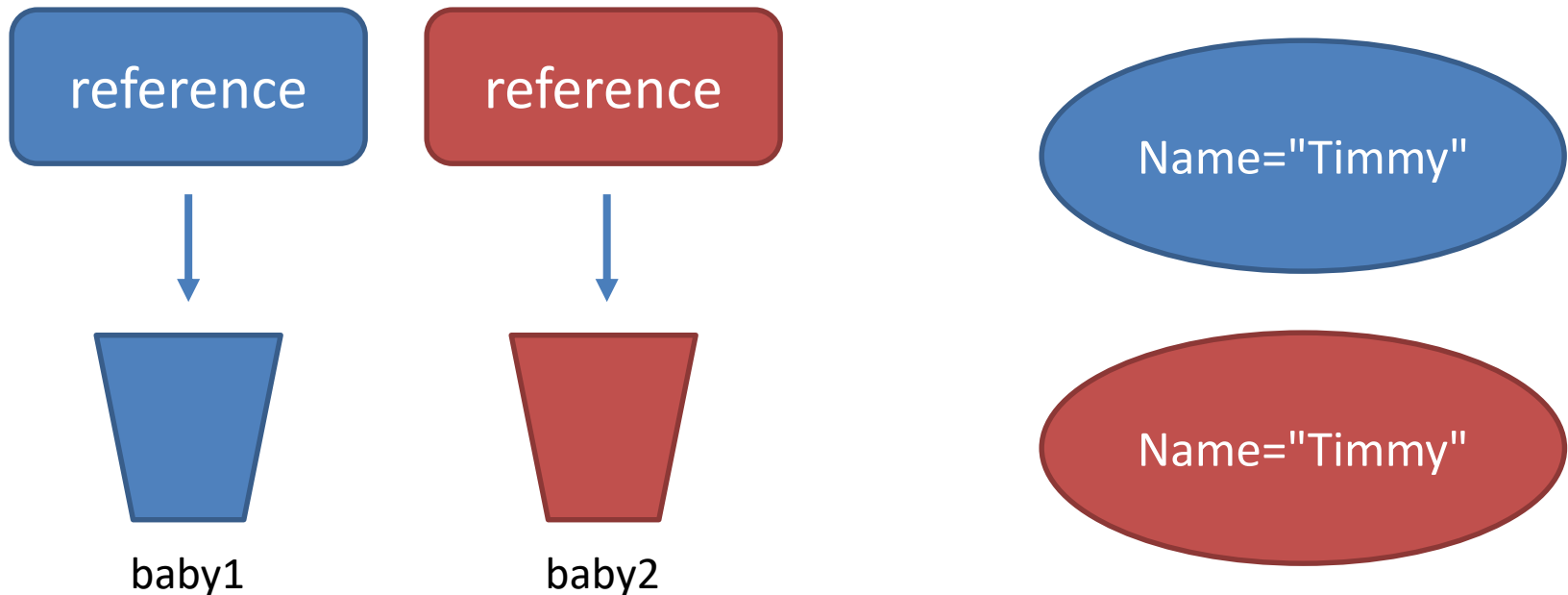
```
Baby tim1 = new Baby("Timmy");  
Baby tim2 = new Baby("Timmy");  
// true or false?  
boolean b = (tim1 == tim2);
```



# References

- Using `=` updates the reference.

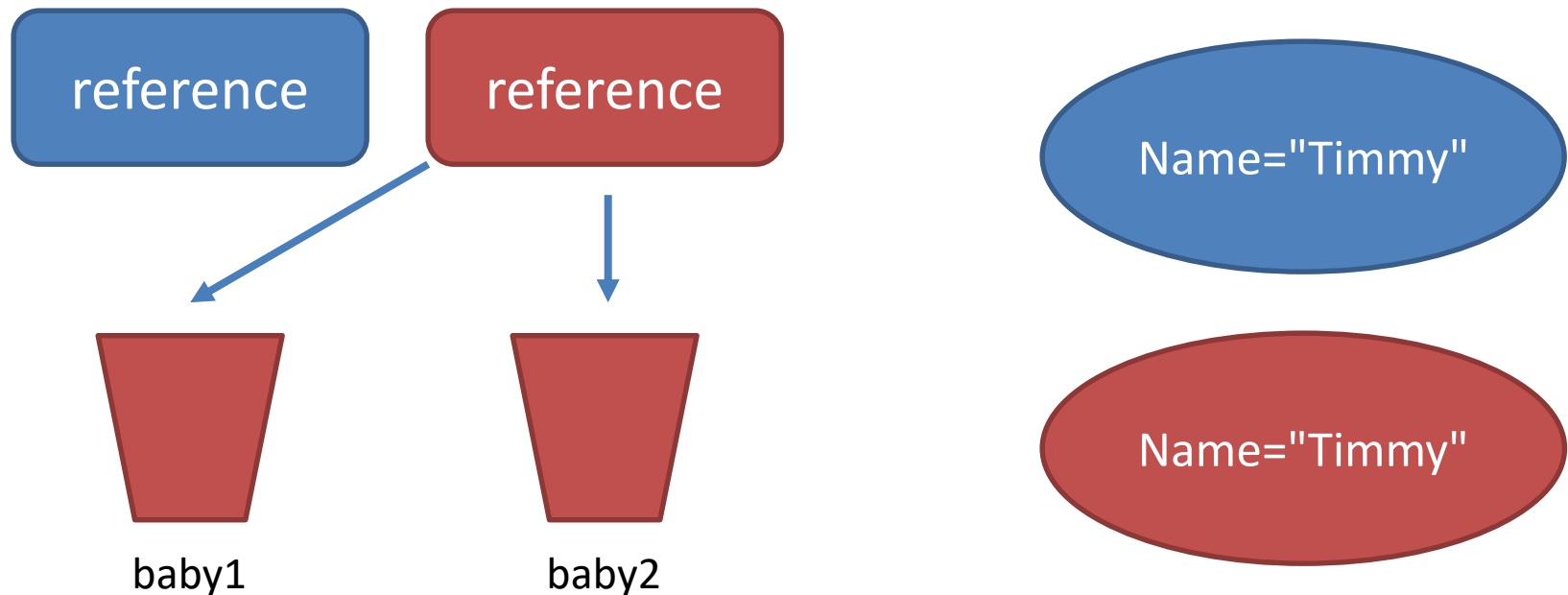
`baby1 = baby2;`



# References

- Using `=` updates the reference.

`baby1 = baby2;`





# static keyword

- Applies to fields and methods
- `static` means the field/method
  - Belongs to the static context
  - Is defined for the entire class
  - Is not unique to each instance

# static example

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

```
public class Baby {  
    // fields  
    static int babiesMade = 0;  
  
    // constructor(s)  
    public Baby(String n, boolean m) {  
        name = n;  
        isMale = m;  
        babiesMade++;  
    }  
  
    // methods  
}
```

## BankAccount example

```
public class BankAccount {  
    double balance;  
    int transactions;  
    public BankAccount(double initial) {  
        this.balance = initial;  
        this.transactions = 1;  
    }  
  
    public void deposit(double amount) {  
        balance += amount;  
        transactions++;  
    }  
  
    public void withdraw(double amount) {  
        balance -= amount;  
        transactions++;  
    }  
  
    public void monthlyFee() {  
        this.withdraw(10);  
    }  
}
```

# this keyword

- You create a bank account:

```
BankAccount myAcc = new BankAccount(500);
```

- You withdraw \$50:

```
myAcc.withdraw(50);
```

- Here, you specify the method to call and the instance to which the method belongs.

```
public void monthlyFee() {  
    withdraw(10);  
}
```

- What's the instance? → The current instance

# this keyword

```
public void monthlyFee() {  
    this.withdraw(10);  
}
```

**this** keyword refers to the current instance.