CENG 1004
Introduction to Object Oriented Programming

Spring 2016

Özgür Kılıç
Office: E1-03
Office hours: Thu 13:30-15:20
email: ozgur.kilic10@gmail.com

Course Web Page:
piazza.com/mu.edu.tr/spring2016/ceng1004/home

Sign Up Link
piazza.com/mu.edu.tr/spring2016/ceng1004

Today's Topics

- Lecture 2 Review
- Popular Issues
- Object oriented programming
- Defining Classes
- Using Classes
- References vs Values
- Static fields and methods

Lecture 2 Review

The while operator

```
while (condition) {
 statements
}
```

The while operator

```
int i= 0;
while(i < 3){
   System.out.println("Rule #" + i);
   i = i+1;
}
```

- Count carefully
- Make sure that your loop has a chance to finish.

The for operator

```
for(initialization;condition;update){
     statements
}
```

The for operator

```
for(int i = 0; i <3; i=i+1){
    System.out.println("Rule #" + i);
}
```

Note: i = i+1 may be replaced by i++

Embedded loops

```
for (int i = 0; i < 3; i++) {
   for (int j = 2; j < 4; j++) {
        System.out.println (i + “ “ + j);
   }
}
```

do-while statement

- the statements within the do block are always executed at least once

```
do {
    statement(s)
} while (expression);
```

Creating, Initializing, and Accessing an Array

The index starts at zero and ends at length-1.

Example:

```
int[] values = new int[5];
values[0] = 12; // CORRECT
values[4] = 12; // CORRECT
values[5] = 12; // WRONG!! compiles but
                // throws an Exception
                // at run-time
```

Creating, Initializing, and Accessing an Array

Curly braces can be used to initialize an array.

It can ONLY be used when you declare the variable.

```
int[] values  = { 12, 24, -23, 47 };
```

The length variable

Each array has a `length` variable built-in that contains the length of the array.

```
int[] values =  new int[12];
int size =      values.length; // 12

int[] values2 = {1,2,3,4,5}
int size2 =     values2.length; // 5
```

Arrays as parameters

The main method accepts a single argument: an array of elements of type String.

```
public static void main (String[] arguments){
    System.out.println(arguments.length);
    System.out.println(arguments[0]);
    System.out.println(arguments[1]);
}
```

Command-line argument

- Command-line arguments let users affect the operation of the application without recompiling it.

```
java MyApp arg1 arg2
```

Multidimensional Arrays

- You can also declare an array of arrays by using two or more sets of brackets, such as String[][] names.

```
String[][] names = {
        {"Mr. ", "Mrs. ", "Ms. "},
        {"Smith", "Jones"}
};

// Mr. Smith
System.out.println(names[0][0] + names[1][0]);

// Ms. Jones
System.out.println(names[0][2] + names[1][1]);
```

Recursion

- A method of defining a function in terms of its own definition
- Example: the Fibonacci numbers

  $f(n) = f(n-1) + f(n-2)$

  $f(0) = f(1) = 1$ <<<<< Base Case

- In programming recursion is a method call to the same method. In other words, a recursive method is one that calls itself.

Recursion

- Definition of factorial:

$$n! = n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1$$

- Recursive definition:

$$n! = \begin{cases} n \cdot (n-1)!, & n > 0 \\ 1, & n = 0 \end{cases}$$

Recursion

```
public static int factorial(int n) {
    if (n == 0)
        return 1;
    else{
        return n * factorial(n - 1);
    }
}
```

Questions from last lecture?

Popular Issues

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]`

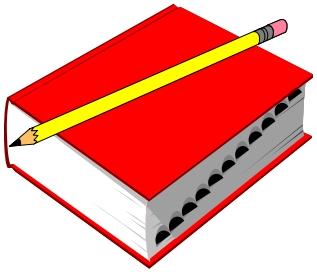
Popular Issues 4

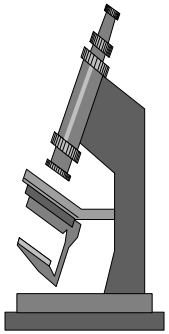
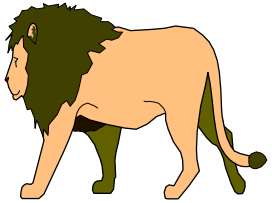
Defining a method inside a method

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

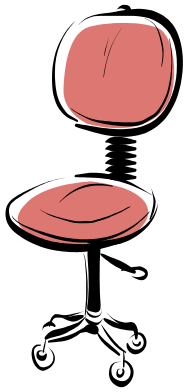
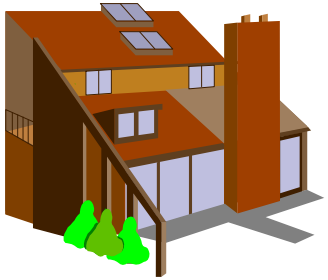
Object oriented programming

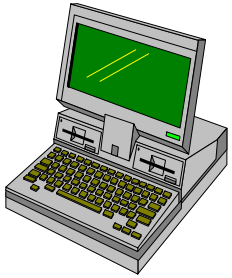
Real World Objects



Real world entities




Real World Objects

Objects have attributes (state)...

ATTRIBUTES
Name : Pamuk
Color : White
Breed : White Terrier
Hungry: Yes

ATTRIBUTES
Current Gear: 4
Current Direction: West
Current Speed: 90 km/h
Color:White

Objects have behaviours

BEHAVIOUR
Barking
Fetching
Eating
Running

ATTRIBUTES
Name : Pamuk
Color : White
Breed : White Terrier
Hungry: Yes

BEHAVIOUR
Change Gear
Change Direction
Accelerate
Apply Brakes

ATTRIBUTES
Current Gear
Current Direction
Current Speed
Color

Objects have behaviours

ATTRIBUTES
On: Yes

BEHAVIOUR
Turn On
Turn Off

ATTRIBUTES
On: Yes
Current
Volume: 5
Current
Station: 103.1

BEHAVIOUR
Turn On
Turn Off
Increase Volume
Decrease
Volume
Seek
Scan
–

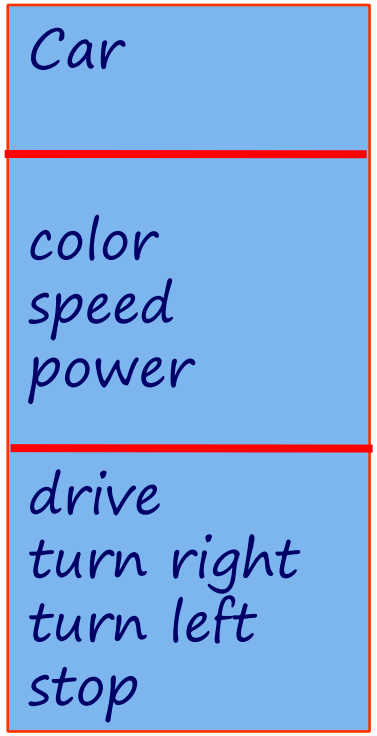
Example: A “Rabbit” object

- You could (in a game, for example) create an object representing a rabbit
- It would have data:
  - How hungry it is
  - How frightened it is
  - Where it is
- And methods:
  - eat, hide, run, jump

Classes (Categories)

Classes

- Serves as template/blueprint from which objects can be created
- Can be used to *create* objects
- Objects are the instances of that class
- Defines attributes and operations



Example: Student

- Represent the real world

  Student

Example: Student

- Represent the real world

Student

name
id
year
courses
email

Example: Student

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

Student

```
String name
String id
int year
ArrayList courses
String email
```

Why use classes?

- Why not just primitives?

```
// student Ali
String nameAli;
int yearAli;

//student Mehmet
String nameMehmet
int yearMehmet;
```

Why use classes?

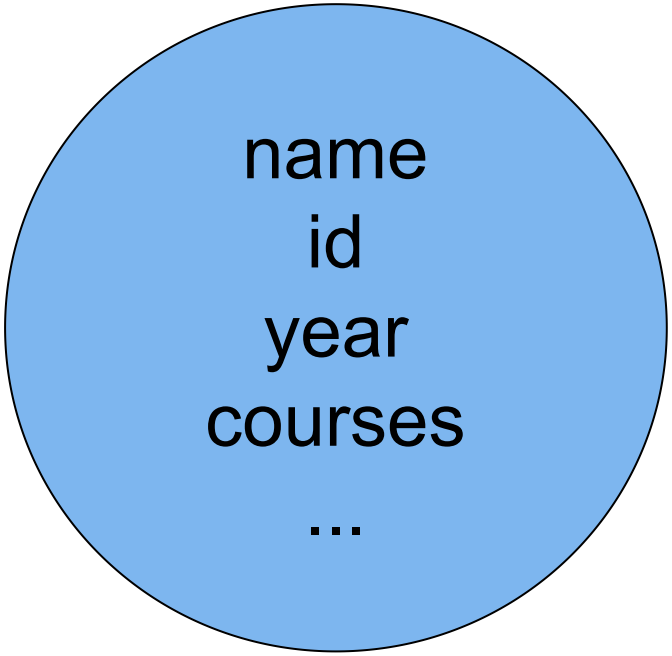
- Why not just primitives?

```
// student Ali
String nameAli;
int yearAli;

//student Mehmet
String nameMehmet
int yearMehmet;
```

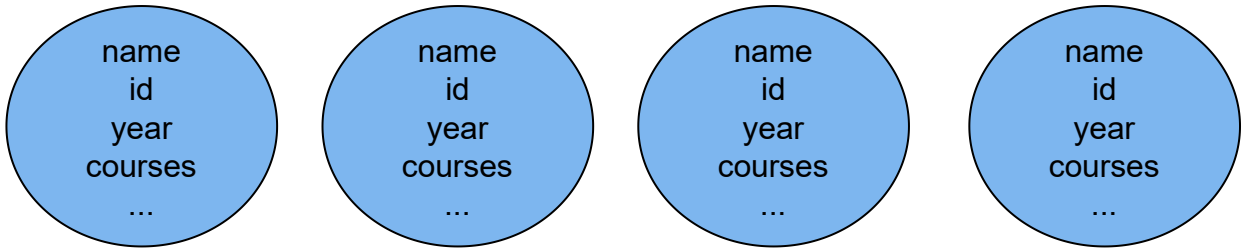
200 Students ?

Why use classes?



Student1

Why use classes?

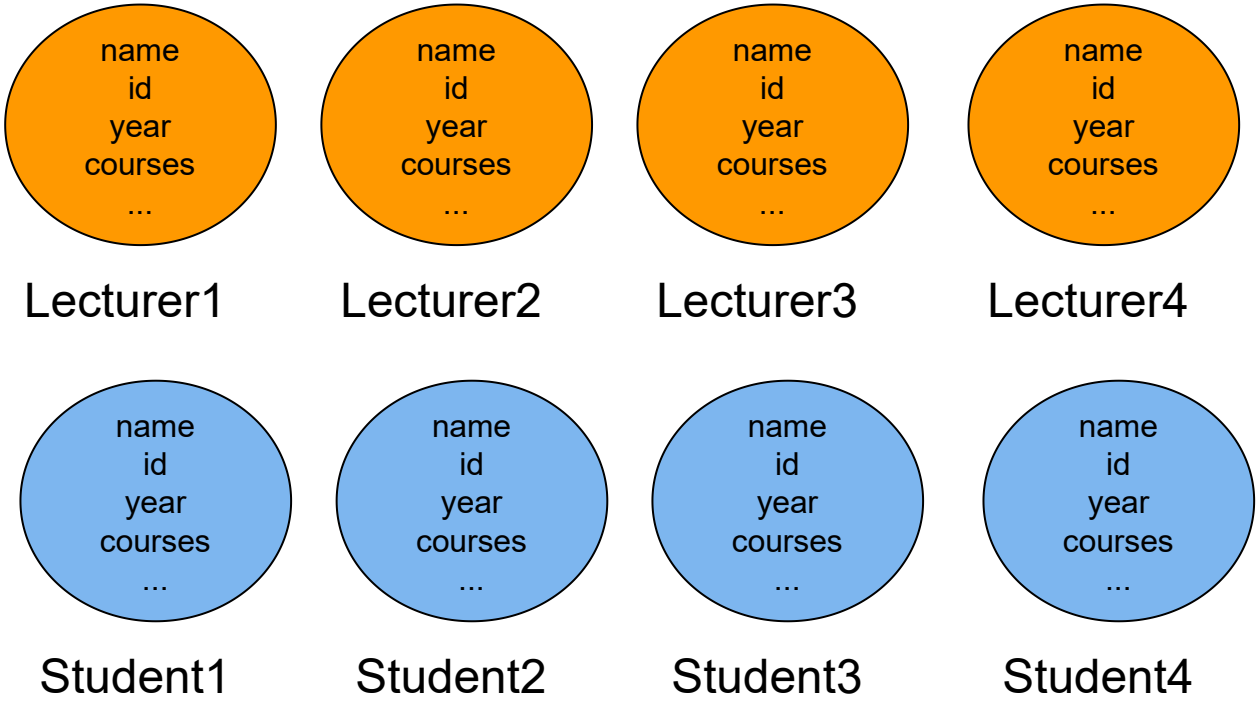


Student1 Student2 Student3 Student4

196
more
Students

Why use classes?

- University

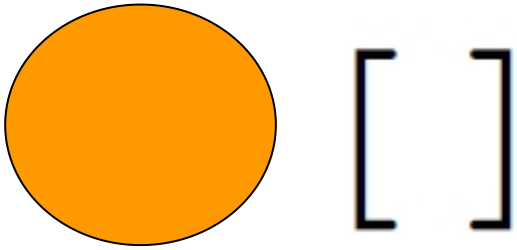


more Lecturers

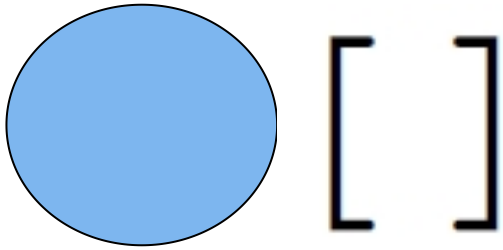
196 more Students

Why use classes?

- University



Lecturer



Student

Defining Classes

Class - overview

```
import java.util.ArrayList;

public class Student {

    String name;
    String id;
    int year;
    ArrayList courses = new ArrayList();
    String email;

    public void registerCourse(String course){
        courses.add(course);
    }

}
```

Class
Definition

Class -overview

```
Student student1 = new Student();
```

Class

Instance

Let’s declare a Student

```
public class Student {

}
```

Let’s declare a Student

```
public class Student {
```


```
}
```

Note

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

Let's declare a Student

```
public class Student {

    TYPE var_name;
    TYPE var_name = some_value;



}
```

Student fields

```
import java.util.ArrayList;

public class Student {

    String name;
    String id;
    int year;
    ArrayList courses = new ArrayList();
    String email;



}
```

Ok, let’s create a student instance!

```
Student student1 = new Student();
```

What about the name of the student?

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 () {
}
```

Student constructor

```
import java.util.ArrayList;

public class Student {

    String name;
    String id;
    int year;

    public Student(String studentName, int studentYear){
        name = studentName;
        year = studentYear;
    }

}
```

Student methods

```
import java.util.ArrayList;

public class Student {

    String name;
    String id;
    int year;
    ArrayList courses = new ArrayList();
    ...
    public void registerCourse(String course){
            courses.add(course);
    }
}
```

Student Class

```
import java.util.ArrayList;

public class Student {

    String name;
    String id;
    int year;
    ArrayList courses = new ArrayList();
    String email;

    public void registerCourse(String course) { ... }
    public void unregisterCourse(String course) { ... }
    public void incrementYear() { ... }

}
```

Class
Definition

Using Classes

Classes and Instances

```
import java.util.ArrayList;

public class Student {

    String name;
    String id;
    int year;
    ArrayList courses = new ArrayList();
    String email;

    public static void main(String[] args){

        Student student1 = new Student("Ali", 1);
        Student student2 = new Student("Mehmet", 3);
    }

    public Student(String studentName, int studentYear){
        name = studentName;
        year = studentYear;
    }
}
```

Accessing fields

- Object.FIELDNAME

```
Student student1 = new Student("Ali", 1);

System.out.println(student1.name);
System.out.println(student1.year);
```

Calling Methods

- Object.METHODNAME([ARGUMENTS])

```
Student student1 = new Student("Ali", 1);

student1.incrementYear();
student1.registerCourse("CENG 1004");
```

References vs Values

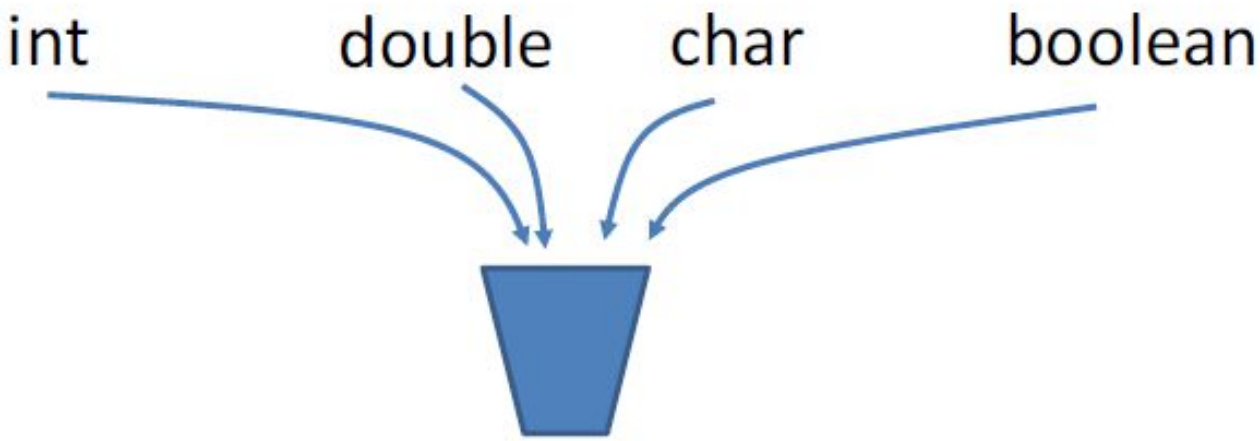
Primitives vs References

- 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

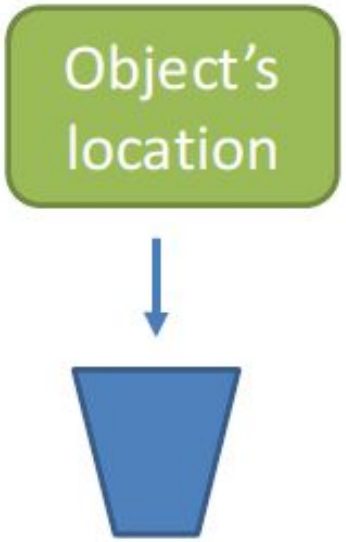


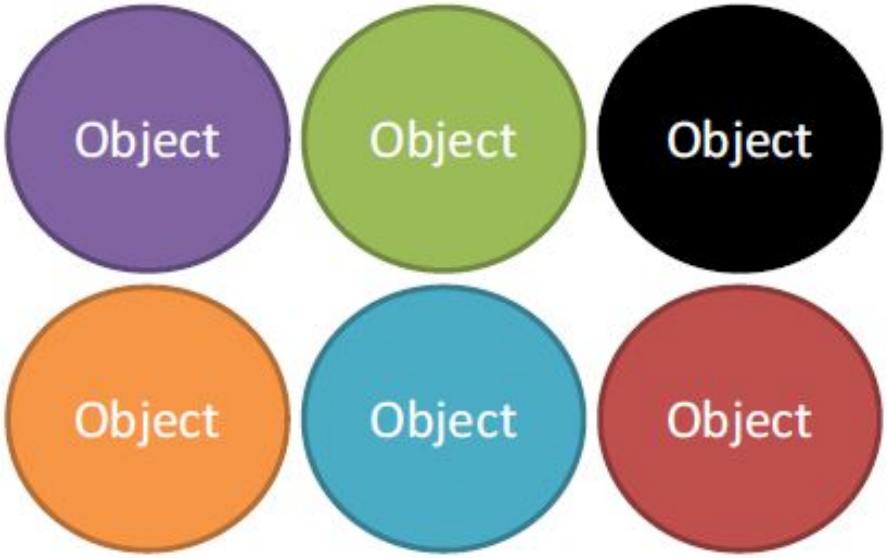
How java stores objects

- 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

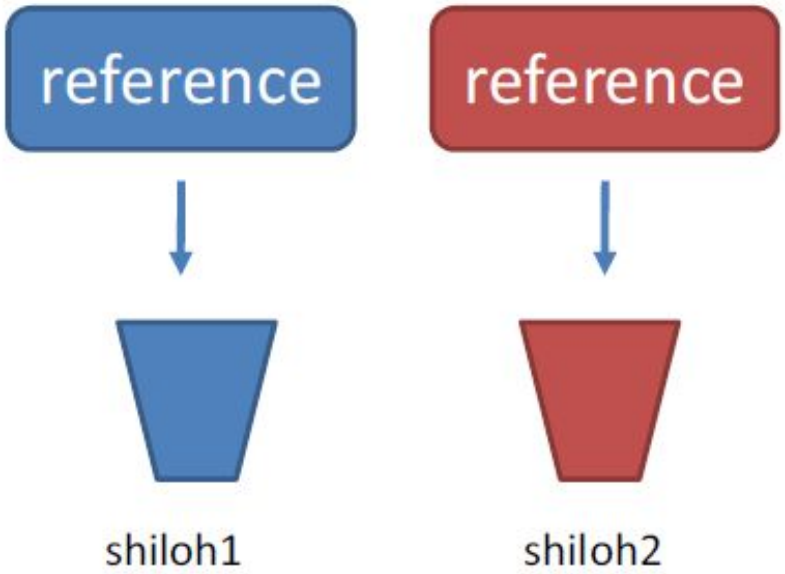
```
Baby shiloh1 = new Baby("shiloh");
Baby shiloh2 = new Baby("shiloh");
```

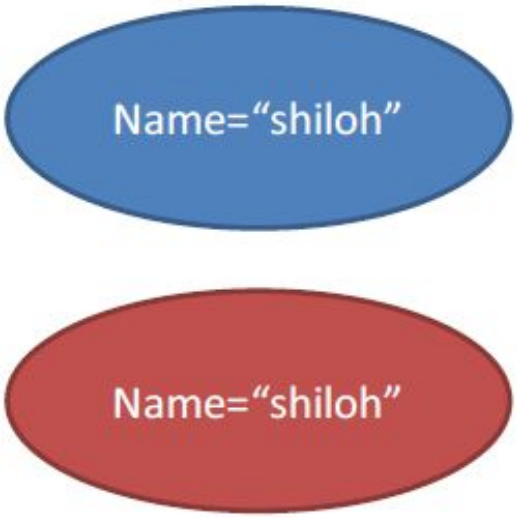
Does `shiloh1 == shiloh2`?

no

References

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



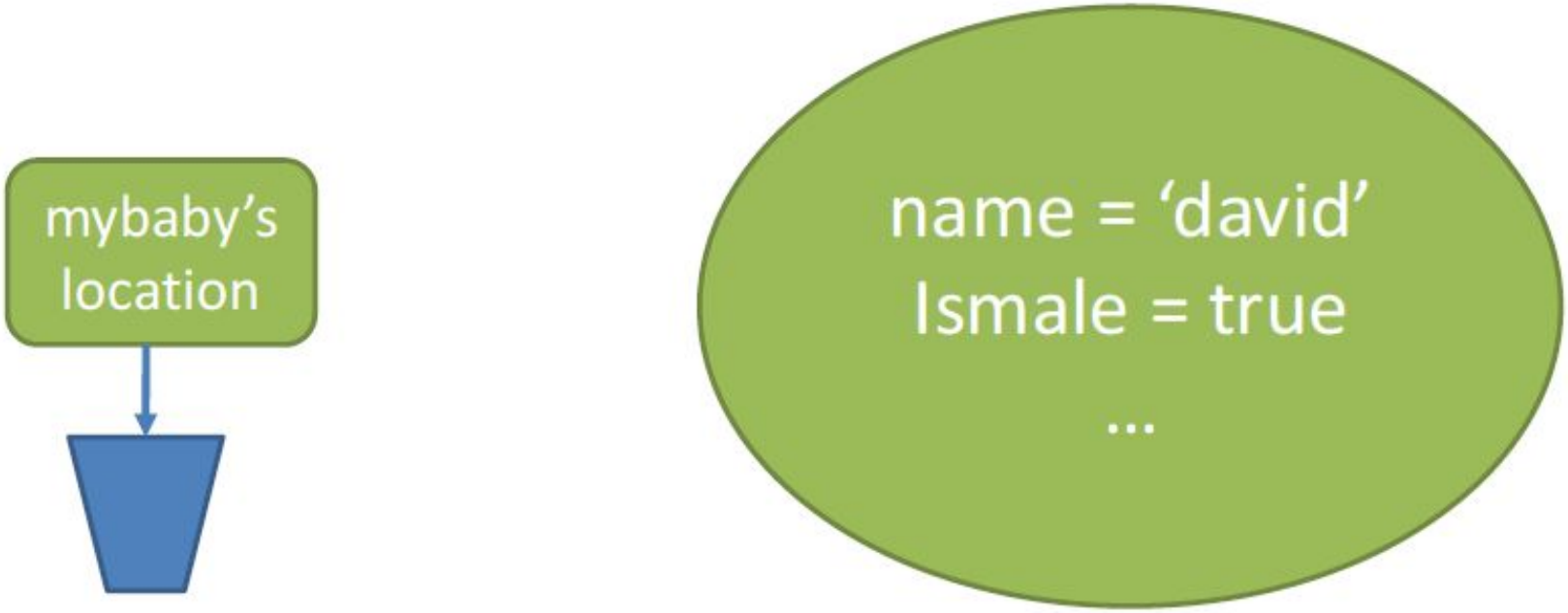


References

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

References

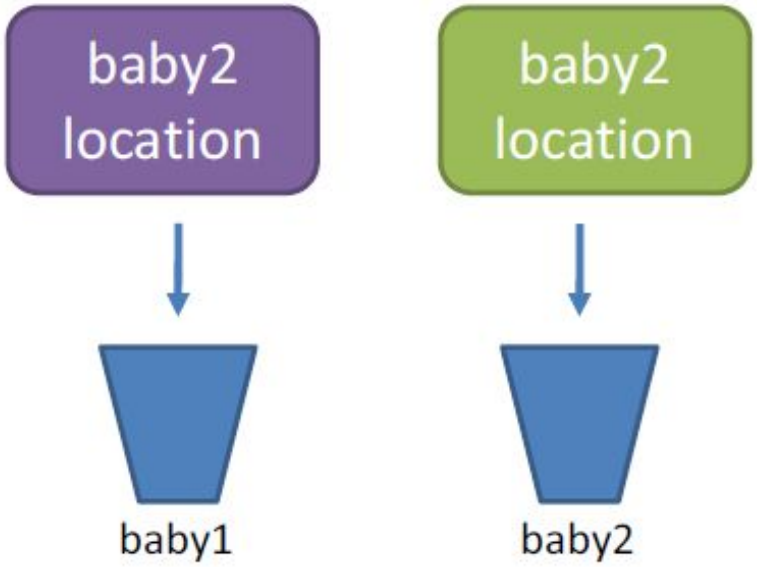
```
Baby mybaby = new Baby('davy', true)
mybaby.name = 'david'
```

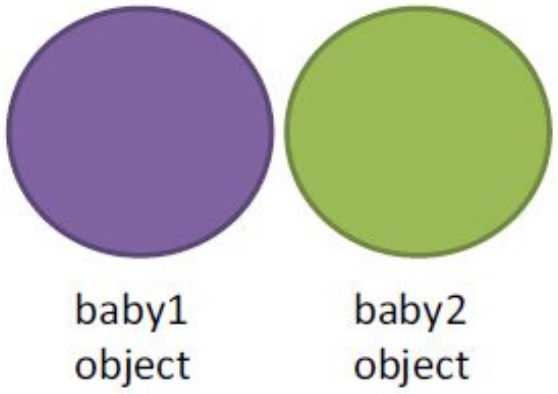


References

- Using = updates the reference.

```
baby1 = baby2
```

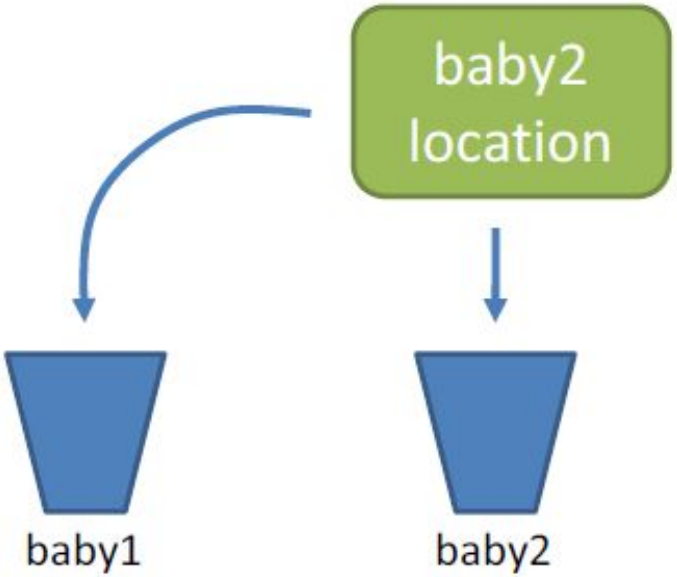


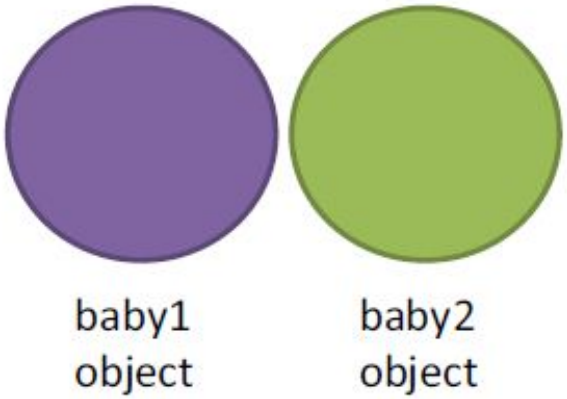


References

- Using = updates the reference.

```
baby1 = baby2
```





static fields 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 Student {

    String name;
    int year;

    static int count = 0;

    public static void main(String[] args){
        Student.count = 50;
        Student student1 = new Student("Ali", 1);
        Student student2 = new Student("Mehmet", 3);
        Student.count = 2;
    }

    public Student(String studentName, int studentYear){
        name = studentName;
        year = studentYear;
    }
}
```

static example

- Keep track of the number of students

```
public class Student {

    String name;
    int year;

    int count = 0;

    public Student(String studentName, int studentYear){
        name = studentName;
        year = studentYear;
        count++;
    }
}
```

static field

- Keep track of the number of students

```
public class Student {

    String name;
    int year;

    static int count = 0;

    public Student(String studentName, int studentYear){
        name = studentName;
        year = studentYear;
        count++;
    }
}
```

static method

- Keep track of the number of students

```
public class Student {

    String name;
    int year;

    static int count = 0;

    public static double calculateAverageGrade (Student[] students){
            ...
    }
}
```

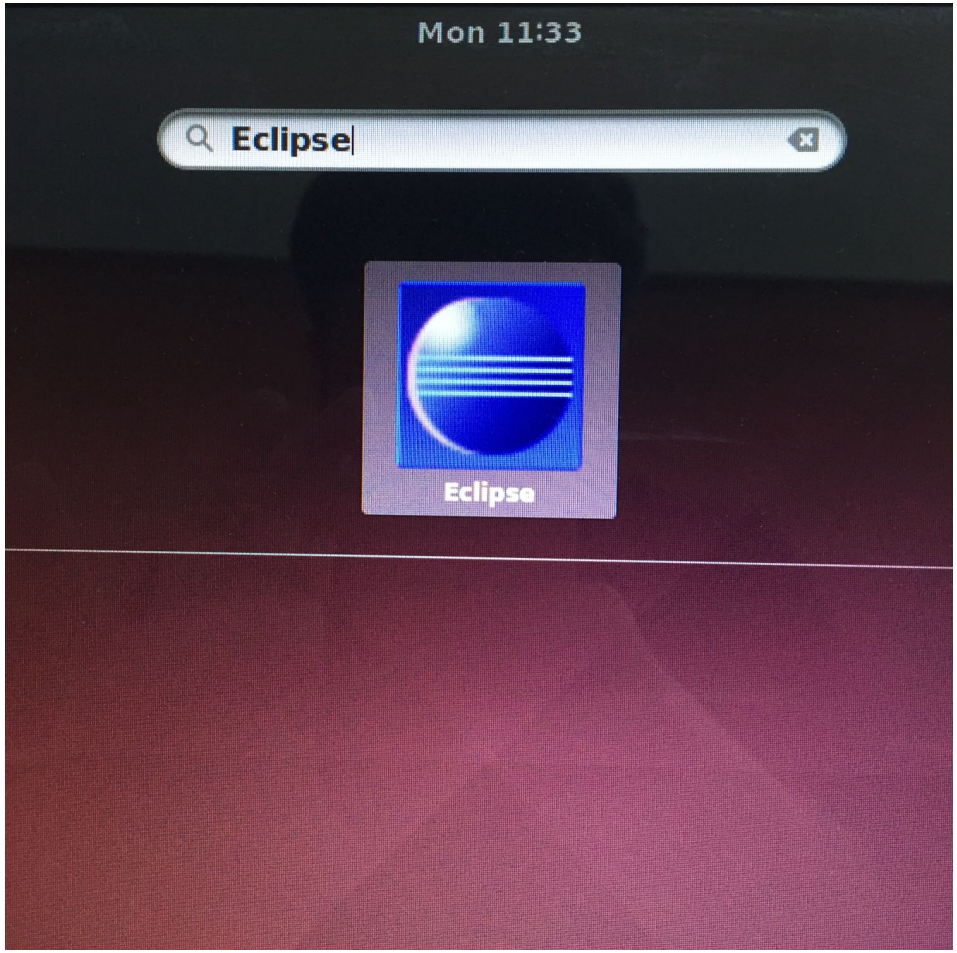
Summary for today

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

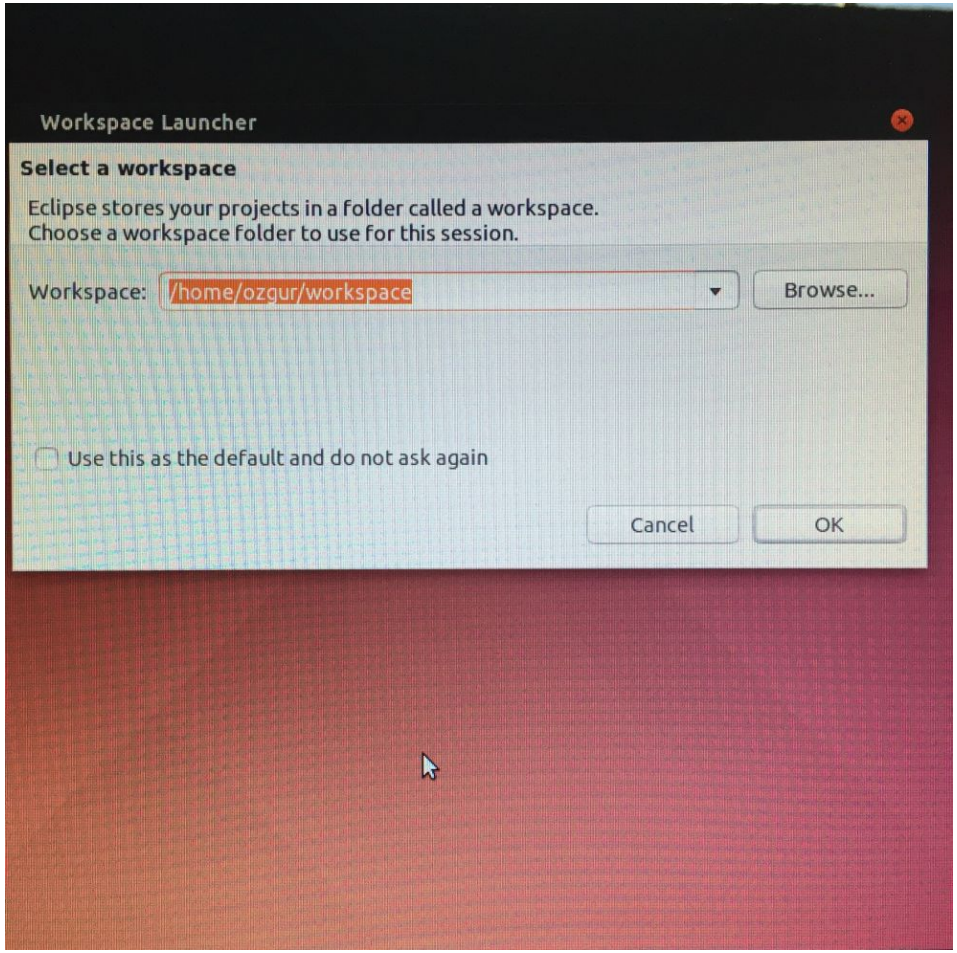
Before Lab

- If you use laptop in lab hours
  - install Eclipse IDE for Java Developers
    - http://www.eclipse.org/downloads/packages/release/Mars/2-RC3
- Otherwise
  - make sure you have no problem if you launch the Eclipse application installed in computers in the linux lab

Launching Eclipse



Launching Eclipse



References

- http://math.hws.edu/javanotes/
- http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-092-introduction-to-programming-in-java-january-iap-2010/lecture-notes/
- https://docs.oracle.com/javase/tutorial/java
- https://www.cs.upc.edu/~jordicf/Teaching/programming/pdf/IP07_Recursion.pdf