CENG 1004
Introduction to Object Oriented Programming

Spring 2018

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

Course Web Page:
piazza.com/mu.edu.tr/spring2018/ceng1004/home

Sign Up Link
piazza.com/mu.edu.tr/spring2018/ceng1004

Algorithm Development

Algorithm Development

- An algorithm is more like the idea behind the program, but it’s the idea of the steps the program will take to perform its task
- An algorithm can be expressed in any language, including English.
- The steps don’t necessarily have to be specified in complete detail, as long as the steps are unambiguous

Stepwise refinement

- Write a description of the task
- Then take that description as an outline of the algorithm you want to develop.
- After that iteratively refine and elaborate that description, gradually adding steps and detail.
- Continue to do so until you have a complete algorithm that can be translated directly into a program using a programming language.
- This method is called stepwise refinement, and it is a type of top-down design.

Pseudocode

- Algorithms are generally written using pseudocode

- Pseudocode consists of informal instructions that imitate the structure of programming languages without the complete detail and perfect syntax of actual program code.

Example

- Problem Description:
  - Print the Prime numbers that are less than 100

Example: first refinement

```
For each number less than 100
  Check if the number is prime or not;
  If the number is prime
      Print the number;
```

Example: second refinement

```
For each number less than 100

    Let divisor = 2;
    Let isPrime = true;

    While divisor is less than number and isPrime is true
        If the number is divisible by divisor
                isPrime =false;
        increment divisor;

    If the isPrime is true
        Print the number
```

Coding Your Algorithm

- Indent your code, even indent your pseudocode!
- Know the syntax of your language – it will help you work effectively with the compiler.
- In general, when the compiler gives multiple error messages,
  - don’t try to fix the second error message from the compiler until you’ve fixed the first one.
- Take the time to understand the error before you try to fix it.
  - Programming is not an experimental science.

Recursion

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

- To solve a problem recursively
  - break into smaller problems
  - solve sub-problems recursively
  - assemble sub-solutions

- Write a function that computes the sum of numbers from 1 to n

  int sum (int n)

  1. use a loop
  2. recursively

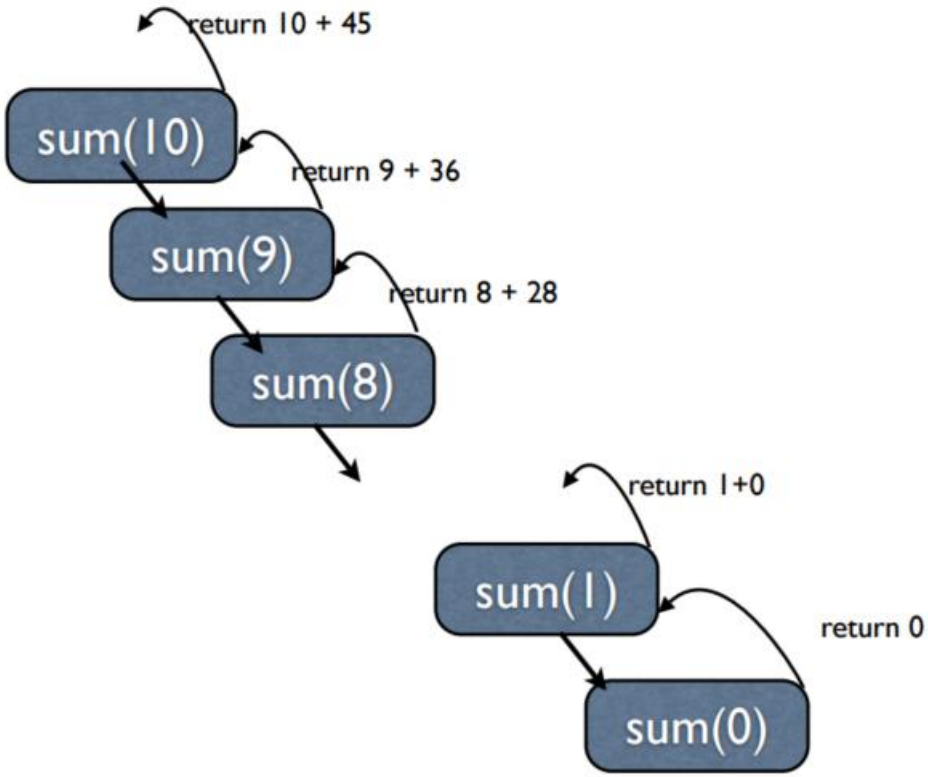
Recursion

```
//with a loop
int sum (int n) {
    int s = 0;
    for (int i=0; i<n; i++)
        s+= i;
    return s;
}
```

```
//recursively
int sum (int n) {
    int s;
    if (n == 0) return 0;
     //else
     s = n + sum(n-1);
     return s;
}
```

How does it work?

Recursion


return 10 + 45
sum(10)
return 9 + 36
sum(9)
return 8 + 28
sum(8)
return 1+0
sum(1)
return 0
sum(0)

Recursion

- Factorial n!

```
public static int factorial(int n) { // iterative solution
        int f = 1;
        int i = 0;
        while (i < n) {
                i = i + 1;
                f = f*i;
        }
        return f;
}
```

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);
    }
}
```

Frequent Issues (I)

- The signature of the main method cannot be modified.

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

}
```

Definition: Two of the components of a method declaration comprise the method signature—the method's name and the parameter types.

Frequent Issues (II)

- Return values: if you declare that the method is not void, then it has to return something!

```
public static int pay(double basePay, int hours){
   if (basePay < 8.0) {
        return -1;
   }else if (hours > 60) {
        return -1;
   }else {
        int salary = 0;
        ...
        return salary;
   }
}
```

Frequent Issues (III)

- Don't create duplicate variables with the same name

```
public static int pay(double basePay, int hours){
   int salary = 0;  // OK
   …
   int salary = 0;  // salary already defined!!
   …
   double salary = 0; //salary already defined!!
   …
}
```

Good Programming Style

Good programming style

The goal of good style is to make your code more readable.

By you and by others.

Rule #1: use good (meaningful) names

```
String a1;
int a2;
double b; // BAD!!

String firstName; // GOOD
String lastName; // GOOD
int temperature; // GOOD
```

Rule #2: Use indentation

```
public static void main (String[] args){
   int x = 5;
   x = x * x;
   if (x > 20) {
       System.out.println(x + “ is greater than 20.”);
   }
   double y = 3.4;
}
```

Rule #3: Use whitespaces

- Put whitespaces in complex expressions:

```
// BAD!!
double cel=fahr*42.0/(13.0-7.0);

// GOOD
double cel = fahr * 42.0 / (13.0 -7.0);
```

Rule #3: Use whitespaces

- Put blank lines to improve readability:

```
public static void main (String[] args){

   int x = 5;
   x = x * x;

   if (x > 20) {
       System.out.println(x + “ is greater than 20.”);
   }

   double y = 3.4;
}
```

Rule #4: Do not duplicate tests

```
if (basePay < 8.0) {
   ...
} else if (hours > 60) {
  ...
} else if (basePay >= 8.0 && hours <= 60){
...
}
```

Rule #4: Do not duplicate tests

```
if (basePay < 8.0) {
    ...
} else if (hours > 60) {
   ...
} else if (basePay >= 8.0 && hours <= 60){
...
}
```

BAD

Rule #4: Do not duplicate tests

```
if (basePay < 8.0) {
   ...
} else if (hours > 60) {
  ...
} else {
...
}
```

Good programming style (summary)

- Use good names for variables and methods
- Use indentation
- Add whitespaces
- Don't duplicate tests

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]`

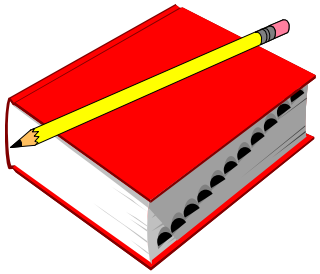
Popular Issues 4

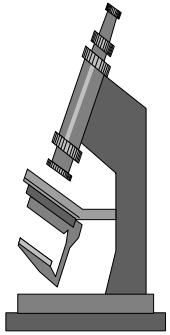
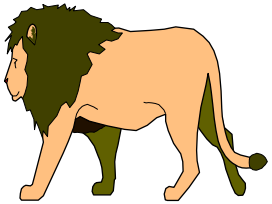
Defining a method inside a method

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

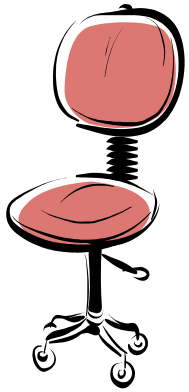
Object oriented programming

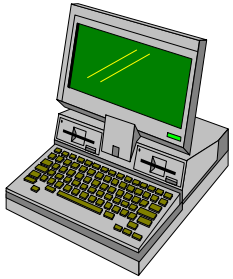
Real World Objects



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:

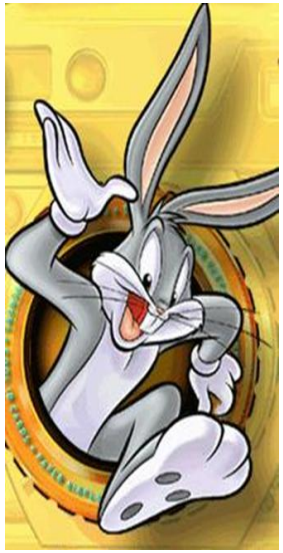
How hungry it is

How frightened it is

Where it is

And methods:

eat, hide, run, jump



Classes (Categories)

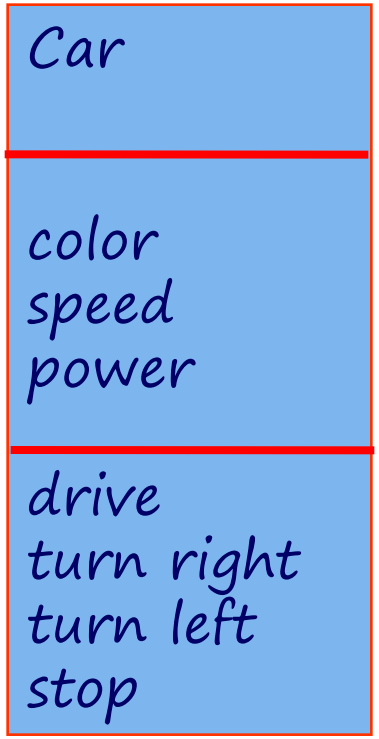
Classes

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?

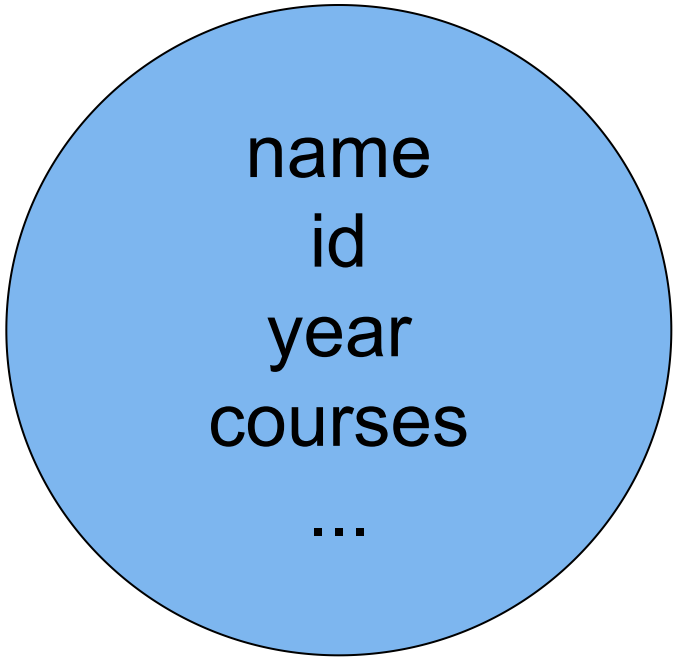
- Why not just primitives?

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

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

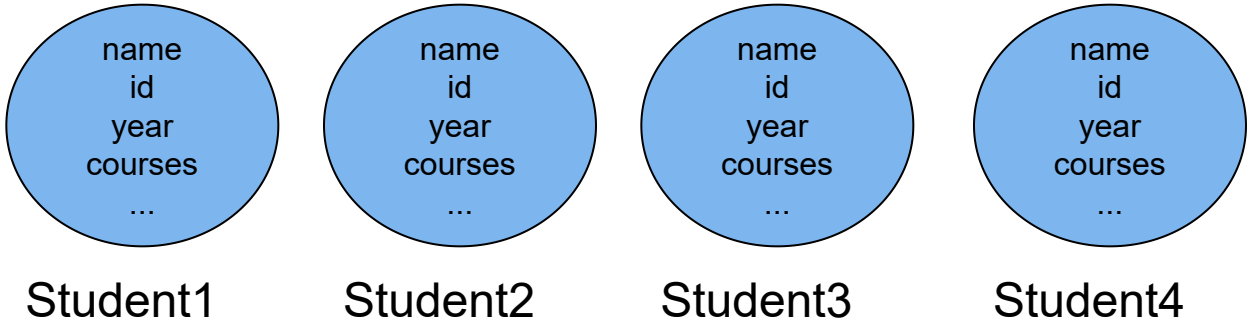
200 Students ?

Why use classes?



Student1

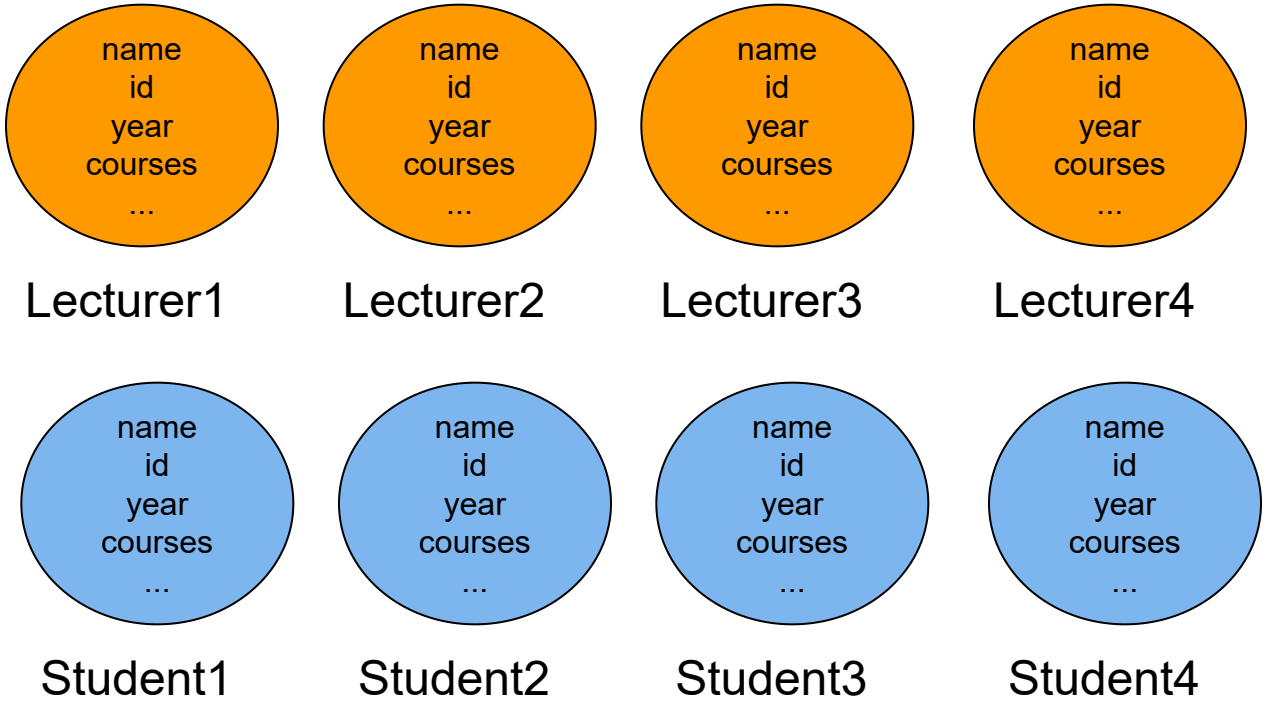
Why use classes?



196
more
Students

Why use classes?

- University

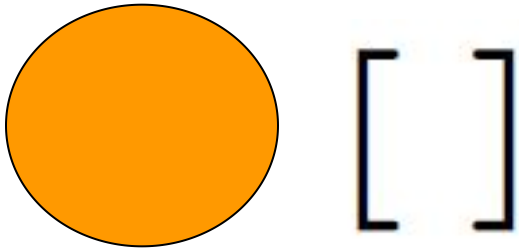


more Lecturers

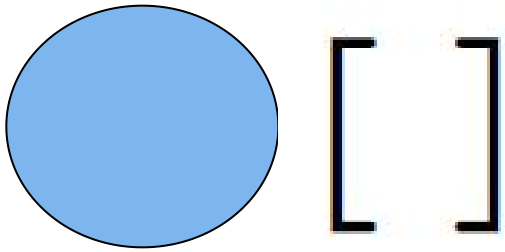
196 more Students

Why use classes?

- University



Lecturer

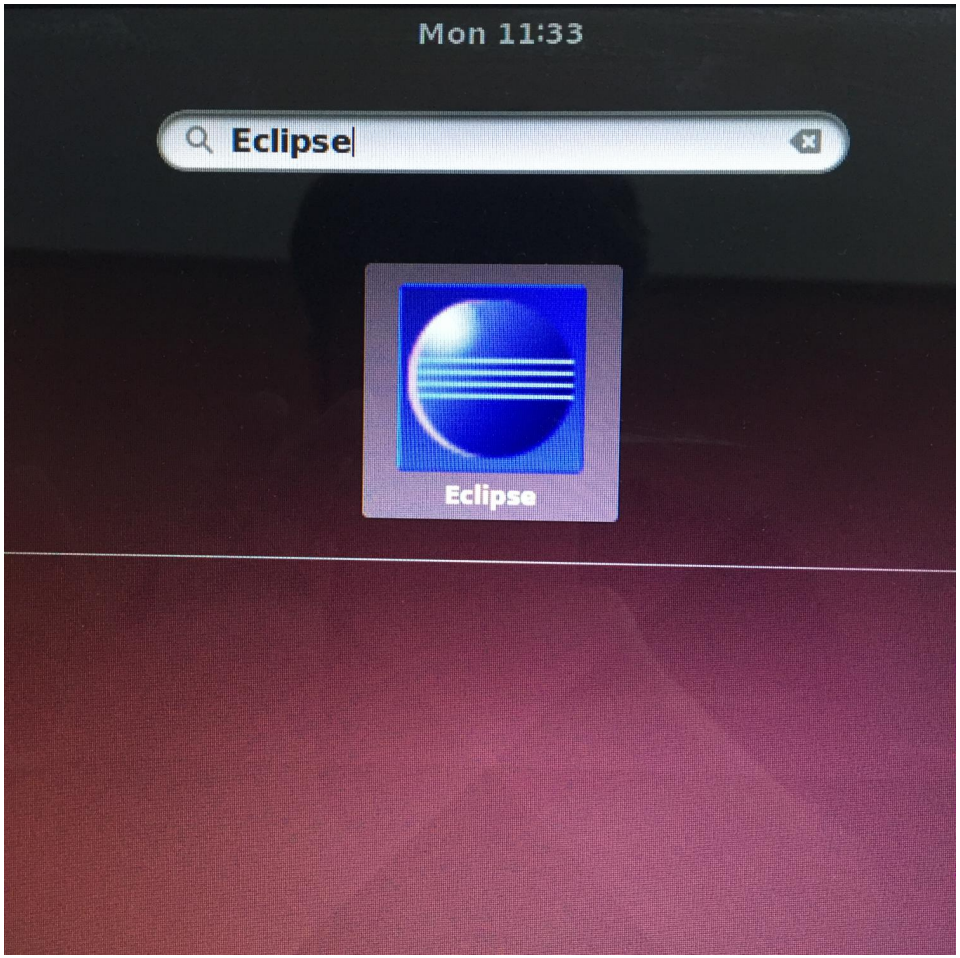


Student

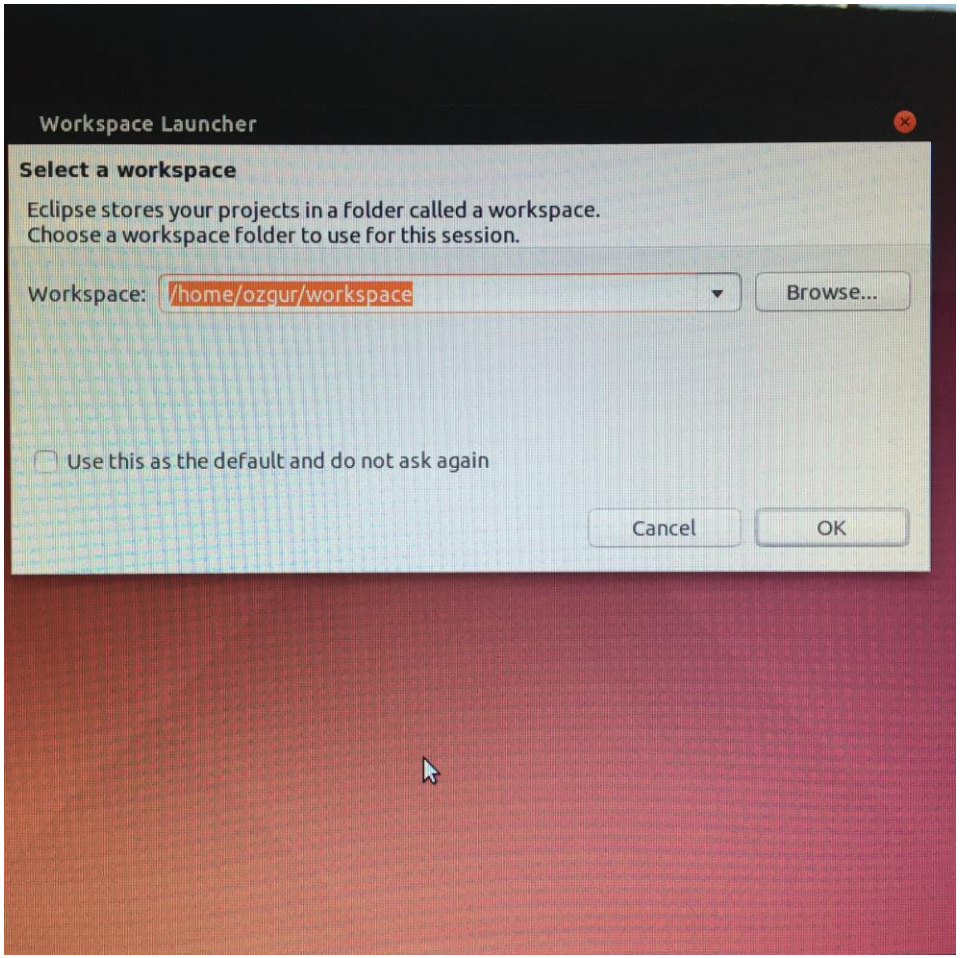
Before Lab

- If you use laptop in lab hours
  - install Eclipse IDE for Java Developers
    - http://www.eclipse.org/downloads/
- 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