

# ESE112

## Java Programming: Intro to OOP

## Object-Oriented Programming (OOP) Model

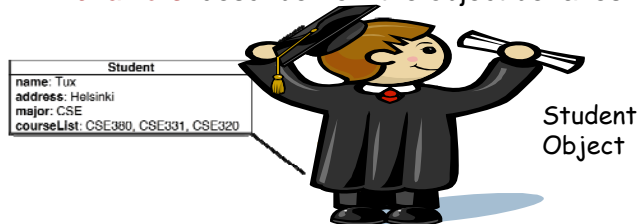
- Design problems/programs such that they correspond to real world *entities*
  - a.k.a *Object*
  - Something that has real existence
  - Examples: person, dog, car, student, bank account , temperature Sensor, frequency generator

ESE112

1

## Object Characteristics

- A Object has
  - *Data/State*: information about that object
  - *Behaviors*: describe how the object behaves



**Data/State**: name, address, major, courseList

**Behavior**: change address, change major, add/drop a course

ESE112

2

## OOP in Java

- A Java program is a **collection** of objects
  - Objects model the parts of a problem
- **Class** in Java for OOP model
  - Is an abstract description of **objects**
    - Describe common features to all objects
    - Templates for creating objects
  - Hence we say that object is an instance of a class
    - Each object has its own unique data

ESE112

3

## Anatomy of Class in OOP

- Classes contain
  - Data Fields that hold the data for each object
    - Data is stored in variables
  - Non-static Methods that describe the actions/behaviors the object can perform
  - Constructor(s) to initialize object with some information

ESE112

4

## OOP Class Structure

```
class Classname {  
    //Data Fields: data for each object  
    ...  
    //Constructor: create a new object of this class  
    ..  
    //Methods: describe the behaviors the object can perform  
    ..  
}
```

ESE112

5

## Data Fields

- Classes describe the data held by each of its objects
  - Also known as *instance* variables

```
class Student {  
    String name;  
    int age;  
    ...rest of the class...  
}
```

Data usually goes first in a class

ESE112

6

## Data Fields contd..

- We can also declare and initialize data fields
  - String name = "unknown";
- However we cannot do:  
String name;  
name = "unknown"; //compiler error

ESE112

7

## Constructor

- Is piece of code (special method) that is executed when an object is created
  - Object created -> means that space is allocated in computers memory to hold information about object
- Most often used to initialize an object's data field's
  - Can initialize data to set value or taken external values
- If you don't write a constructor
  - Java defines one for you (behind the scenes) i.e. *default* constructor
    - The data (if any) will initialize to the default value for that type
    - E.g. for type int, the default value is zero

ESE112

8

## Constructor: Initialize Data Example

Example 1:

```
class Student {
    String name;
    int age;
    //constructor with parameters
    Student(String Name, int Age) {
        name = Name;
        age = Age;
    }
    ...
    //rest of the class
}
```

Example 2:

```
class Student {
    String name;
    int age;
    //constructor without parameters
    Student() {
        name = "Unknown";
        age = 17;
    }
    ...
    //rest of the class
}
```

- Important: Constructor name is same as *ClassName*
- A class can have more than one constructor

ESE112

9

## Creating Objects

- Class is just an abstract description
- In order to use objects we need to create them
- When we create objects, memory is allocated to hold object's data/state
  - This memory is called *heap*
  - Each object gets unique chunk memory to store its data/state
    - Unlike the stack (where method input variables or local variables are place), data on heap is not discarded until forced

ESE112

10

## Creating Objects

- Create object with keyword *new* and call to the *constructor*

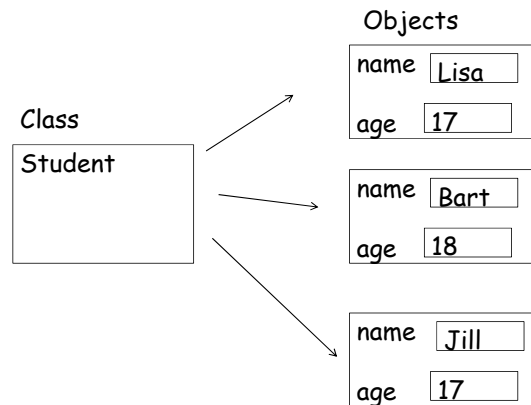
```
new Student();           new Student("Lisa", 5);
  ↑                       ↑
make a new object       make a new object
```

- The keyword *new* allocates space for the object in computers heap memory
- Constructor initializes the data of the object

ESE112

11

## Object is instance of Class



## Referring to the object

- To refer (or access) object's data in memory after it has been created we probably need to access it again
- **Declare** a variable of appropriate *type* to refer to the object
  - The *type* of the variable is the *class* that describes that object
- E.g. Student object we need a variable of *type Student*  
Student s1;
- Then we do:  
s1 = new Student();
- We can also do this in one statement:
  - Student s1 = new Student();

ESE112

13

## Methods

- A class may contain *methods* that describe the behavior of objects
- Two kinds of Methods
  - **Query Methods**: ask an object about its state
    - What's your name? Age? Amount in Bank Account?
  - **Command Methods**: change an object's state
    - Withdraw \$100 from my bank account  
⇒ my bank balance changes

ESE112

14

## Example of Methods

Methods usually go after the data & constructor (style rule)

```
class Student {  
    ...  
    void setStudentAge(int StuAge){  
        age = StuAge;  
    }  
  
    int getStudentAge(){  
        return age;  
    }  
}
```

Note: Methods have access to instance variables defined within class (outside of any method)

ESE112

15

## Sending messages to objects

- We don't perform operations on objects, we "talk" to them
  - This is called *sending a message* to the object
- A message looks like this:  
***objectName.method(extra information)***
  - The ***object*** is the thing we are talking to
  - The ***method*** is a name of the action we want the object to take
  - The ***extra information*** is anything required by the method in order to do its job
  - E.g. `s.getAge()` or `s.setAge(20)`

ESE112

16

## Boe-Bot Examples

- Temp Sensor: basic capabilities of a generic temperature sensor
  - `getTempC/F()`
  - `getTempHi/Lo()`
  - Many more listed in TempSensor class in Boe-Bot Java Documentation
- Frequency Generator: Frequency generation based on pulse width modulation
  - `freqout(int frequency, int time)`
  - Many more listed in Freqout class in Boe-Bot Java Documentation

ESE112

17

## Temporary /Local vs. Instance Variables

- **Temporary/local** variables are known
  - From the point of declaration until the end curly brace of the block in which they are declared
  - Cannot use modifier `private` or `public` with these
- In contrast, **instance** variables are
  - Declared outside of any method
  - Known to all methods in the class in which they are declared
  - Can use modifier `private` or `public` with these
    - More on this later

ESE112

18

## OOP Recap

- **Class**: a template for creating objects
  - Variables – data
  - Methods – behavior
  - Constructor – initialize data
- An object is an **instance** of a **class**
  - `Student s = new Student("Lisa",10);` -> `s` is an object of class `Student`
- A Java program is collection of co-operating objects
  - E.g. Lord of the Rings Simulation
    - One Human class, multiple Human objects
    - One Elf class, multiple Elf objects
    - One Orc class, multiple Orc objects
    - One weapon class, multiple weapon

ESE112

19