Week6 Notes

Eclipse IDE and Code Conventions

- The reason why we used text editor so far is to master Java syntax.
- In real life we mostly use IDEs for projects.
- We will use Eclipse as IDE in this course. However it might look slightly different depending on the version or OS. Understand the logic then google any issue you face first.
- First you have to create a java project.
- src: where source files are usually located. You have to locate your .java files here. In general, you need to know where your projects are in your computer and how they are organized.
- Outline shows member functions, variables etc. in a class.
- You can check **conventions** from Oracle's web-site: https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html
- Naming convention:
    - Classes starts with upper-case (e.g. class TwoPeople)
    - Functions and variables: lower-case first word, upper-case other words. (e.g. myVariable, runFast())
    - Constants (i.e. data types that we never change) : upper-case letters (e.g. static final int MIN_WEIGHT=10.)
    - Long names are fine (unless too long). Be crystal clear when you are naming.
- IDEs suggest very good features: auto-complete, warnings, suggestions etc.
- Debug: Super-powerful life saving mode.
- Break points can be interpreted as milestones. When you debug your code, you can define intervals with break points.
- Step into: Goes into the function.
- Step over: Immediately jumps into next line.
- From expressions tab, you can query values, call functions etc. It depends on scope of the debug.

Constructors:

- Initialization of an object: -Classname- -objectname- = new -Classname(-constructor input-)-
- When we use default constructor, it creates objects with default values.
- Syntax (Constructor):
  -Classname(-inputs-){
  this.-variableName- = input //declaring the value of variable
  this.-variableName- = -functionName(input)- //you can call function inside the

constructor and manipulate inputs.
}

- Use “this” for avoiding confusions when dynamically changing variables for an object.
- When you create an object, inputs should be in same order and data-type with how you define the constructor. Example:

  Person(int age, double weight, int height, boolean female){
  // CODE
  }
  Person erhun = new Person(25(int age) , 75.3(double weight), 182(int height), false(boolean female))

- Constructor does not have return type.
- (Recommended) Don’t do anything other than initialization of your object in constructors.

Function Overloading:

- Garbage Collection (deconstruction): Java automatically clean the memory for you. You do not need to worry about. However you can deconstruct any data type manually.
- Overloading means you can define either a constructor or a method multiple times with different type of inputs.
- You can define multiple constructor. (Overloading). Example:

  Person(int age, double weight, int height, boolean female){
  // CODE (first constructor)
  }

  Person( ){
  // CODE (second constructor)
  }

  You can initialize your object in two ways:
  -Person erhun = new Person(25, 75.3, 182, false); OR
  -Person erhun = new Person();

- You can also overload functions. Example:
  public double calculateBmi(){
  // CODE (first function)
  }

  public double calculateBmi(double height, int weight){
  // CODE (first function)
  }

- If you declare that two objects are equal, then you make their pointers equal (i.e. both objects point the exactly same location in memory). When you update one object, both will be updated.

Access Modifiers:

- The access modifiers in Java specifies the accessibility or scope of a field, method, constructor or class.
- When no access modifier is specified, default access modifier is default.
- Access modifiers and their limitations:

Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

Source: net-informations.com

- Getter/Setter functions: When you restrict the access of the variable for any scope, you can access the values with getter functions and update the values with setter functions. You can also specify the scope of getter and setter functions. Example:

```
//returns the value of the variable
public double getWeight(){
return this.weight;
}

//updates the value of the variable
public void setWeight( double weight){
this.weight = weight;
}
```

- setWeight(-double weight-) function does exactly the same thing with updateWeight(-double weight-) function. You do not have to give a specific set or get names. However, we call them setter or getter by convention 🙂