Advanced Object Oriented Programming

EECS2030

Section M

Who Am I?

- Dr. Mufleh Al-Shatnawi
- office
  - Location: TBA
  - hours : 4:00PM -- 5:00PM on Tuesdays and Thursdays; or by appointments
- email
  - **mufleh@eecs.yorku.ca**

Course Format

- Everything you need to know will eventually be on the York University Moodle site.
- York course Moodle [http://moodle.info.yorku.ca/]
- Check this site regularly for announcements and learning resources

Course Description

- While **LE/EECS1020** and **LE/EECS1021** focuses on the client concern, this course focuses on the **concern of the implementer**.
- Hence, **the student will be asked to implement a given API**.

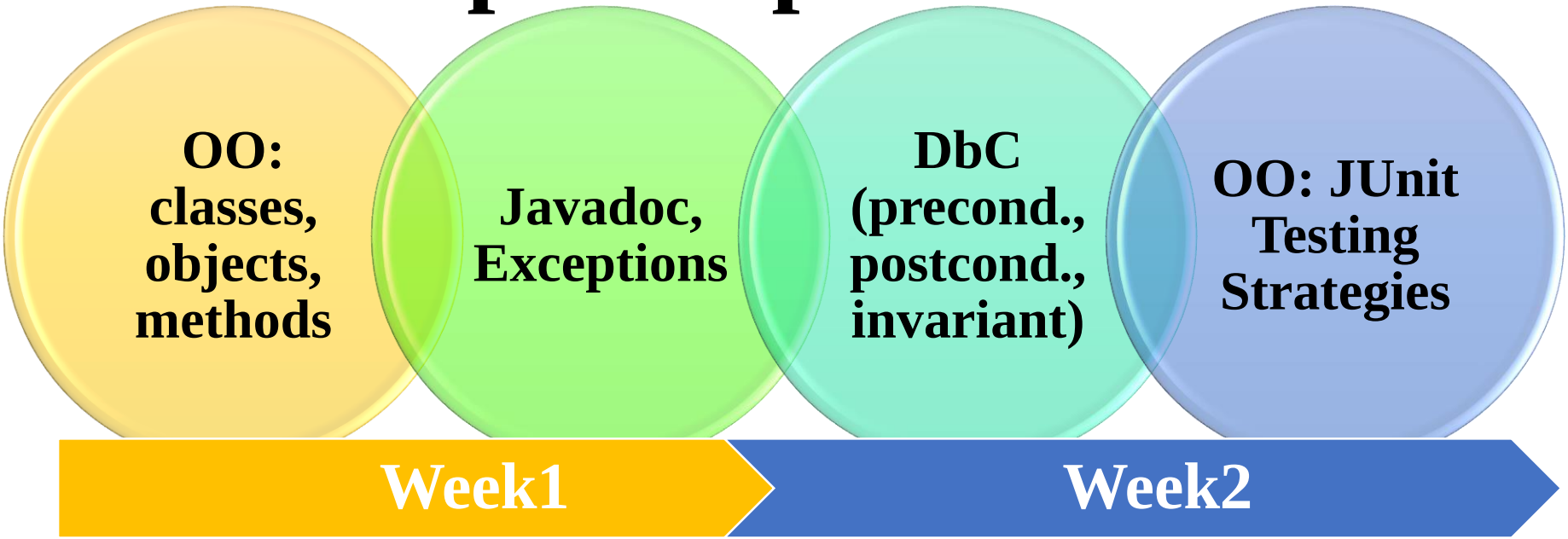
Prerequisites

You must at least have the below prerequisites:

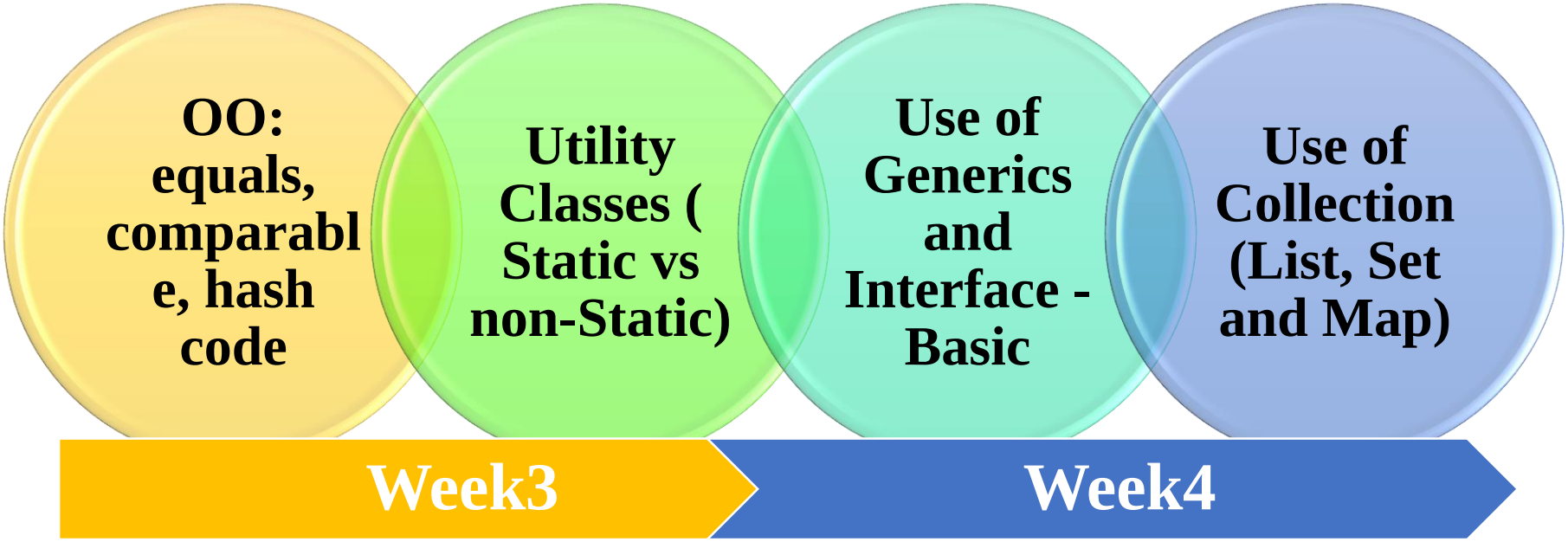
- LE/EECS1021 3.00 or
- LE/EECS 1020 (prior to Fall 2015) 3.00 or
- LE/EECS1022 3.00 or
- LE/EECS 1720 3.00.

The General Prerequisite is a cumulative GPA of 4.50 or better over all major EECS courses.

Course Topic Map

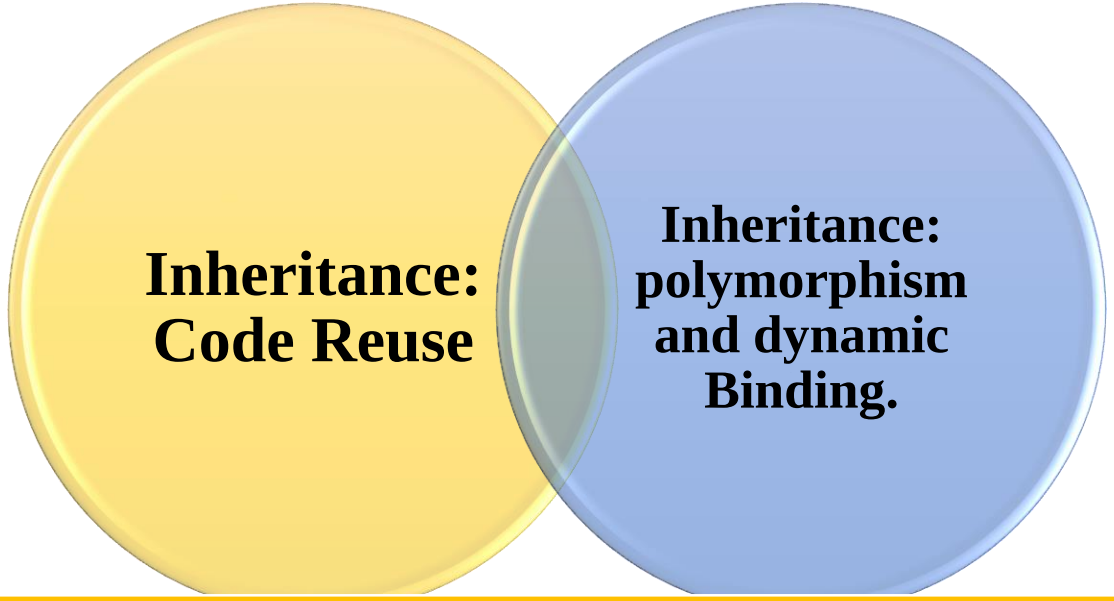


Monday, Jan., 14 - Lab Test 0: Previous Course skills


OO: equals, comparable, hash code
Utility Classes (Static vs non-Static)
Use of Generics and Interface - Basic
Use of Collection (List, Set and Map)
Week3
Week4

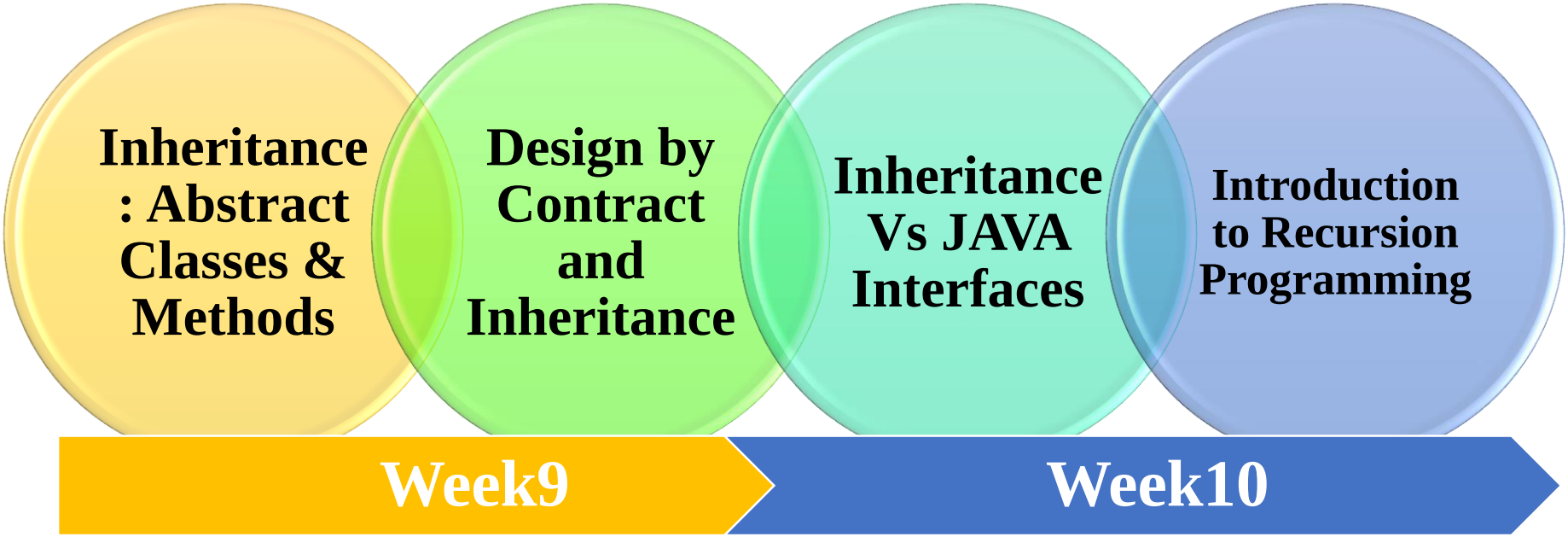
Monday, Feb., 4 - Lab Test 1: Objects, Classes, Methods

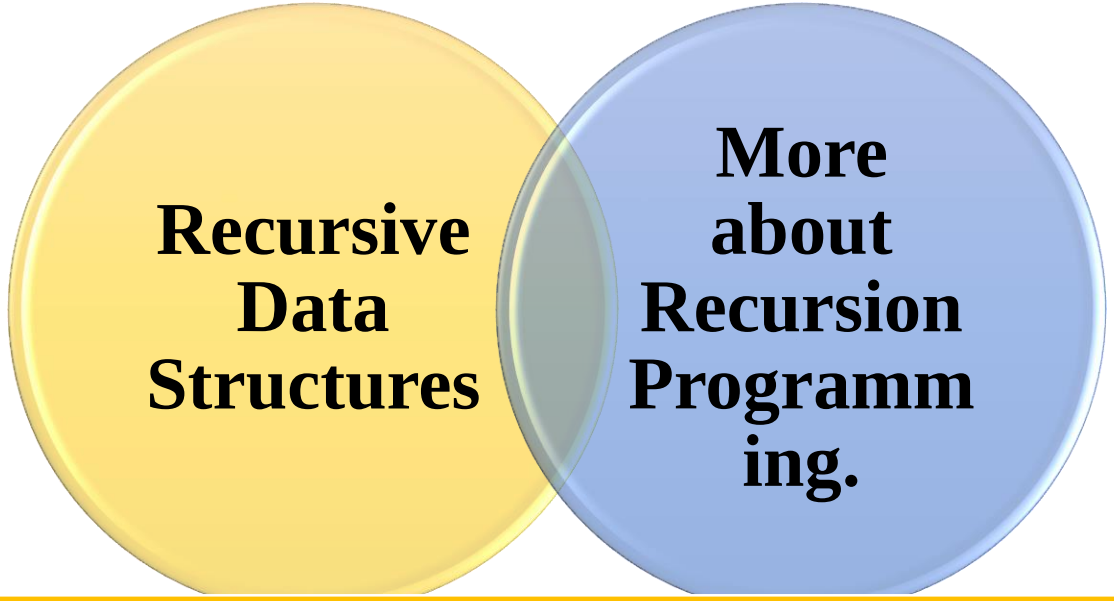
Winter Reading Week (no classes, University open): February 16 - 22



Week8

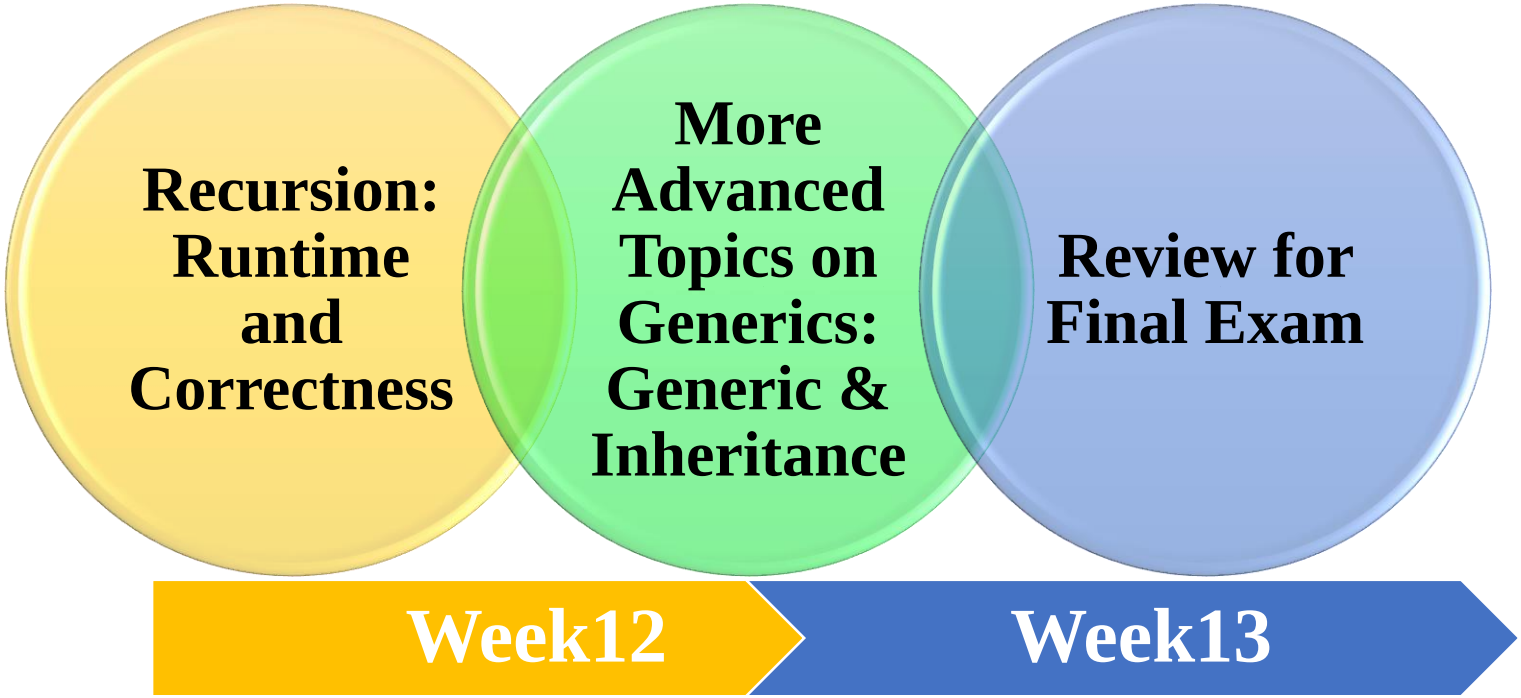
Monday, Mar., 4 - Lab Test 2: Aggregation Composition


Inheritance : Abstract Classes & Methods
Design by Contract and Inheritance
Inheritance Vs JAVA Interfaces
Introduction to Recursion Programming
Week9
Week10



Week11

Monday, Mar., 25 - Lab Test 3: Inheritance and basic Recursion Programming


Recursion: Runtime and Correctness
More Advanced Topics on Generics: Generic & Inheritance
Review for Final Exam
Week12
Week13

Lectures

- One person talks at a time in class, please!
- Do not use Cell phones during lectures.

- Attendance to all lectures and labs are necessary.
  - Don’t consider coming to classes as pressure.

Labs

- In Prism computing labs (LAS1006)
- Lab Zero: due date on **Jan. 10 2019**
  - self-guided, can be done anytime before the start of Week 2
  - using the Prism lab environment
  - using eclipse
  - review previous Java programming skills

- Labs (<=7) consist of a different set of programming problems for each lab
  - *It is expected that you know how to use the lab computing environment.*
- Fun, hard work, a great learning experience

Labs

- Lab Times
  - **LAB (Section-01): Mondays: 10:00-11:30 (LAS 1006)**
  - **LAB (Section-02): Mondays: 13:30-15:00 (LAS 1006)**
- group lab work is allowed and strongly encouraged for Labs (not Lab Zero)
  - groups of up to size 3
  - see *Academic Honesty* section of syllabus
    - <u>Do not submit work that is not wholly your own</u>
    - It is absolutely not acceptable that you:
      - share your (programming or written) solutions with others;
      - copy and paste solutions from elsewhere and claim that they are yours.

Labs

- Tips for effective group work
  - alternate who is doing the typing (the *driver*) every few minutes
    - **don't allow the stronger programmer to do everything** !!
    - if you are the stronger programmer then try explaining your thought processes to your group partners
- if you aren't typing then you are a *navigator* you should be:
  - watching what the driver is doing to catch mistakes
  - planning what the group should do next
  - **developing test cases** to test the code that is being written

EECS Account needed for EECS2030

- **You must at least have a Prism Lab account before starting the course.**
  - If you don't have a Prism Lab account for EECS courses, please consult lab monitors at **LAS 1006**.
  - You must have a **YoRkU ID** card with you at all times.
  - If you don’t have the account ready, you cannot do the labs and tests.

Lab Tests

- Computer test, based on lab exercises and lecture materials
- Each lab section has its own lab tests

Tests

- All testing occurs during your regularly scheduled lab using the EECS **labtest environment**

Test	Weight
Lab Test 0	5%
Lab Test 1	15%
Lab Test 2	15%
Lab Test 3	15%
Exam	40%

- miss a test for an acceptable reason?
  - see *Evaluation: Missed tests* section of syllabus

Textbook

- A set of freely available electronic notes is available from the Moodle site
- Recommended textbooks
  - *Building Java Programs*, 4th Edition, S Roges and M Stepp
  - *Introduction to Programming in Java*, 2nd Edition, R Sedgewick and K Wayne
    - does not cover inheritance
  - *Absolute Java*, 6th Edition, W Savitch
- Recommended references
  - *Java 8 Pocket Guide*, Liguori and Liguori
  - *Effective Java*, 3rd Edition, J Bloch

Need Accommodation for Tests/Exams?

- Please approach me (email, in person) as soon as possible, so we can make proper arrangements for you.

Prepare your own machine

- Install Java, and Eclipse
  - http://www.oracle.com/technetwork/java/javase/overview/index.html
  - https://www.eclipse.org/downloads/ **[Eclipse IDE for Java Developers]**

Recommended Online Jackie's tutorials

- **Eclipse**
  - **https://www.eecs.yorku.ca/~jackie/teaching/tutorials/index.html#eclipse**
  - **Importing a Project from an Archive File**
- **Objected Oriented Programming in Java**
  - **https://www.eecs.yorku.ca/~jackie/teaching/tutorials/index.html#oop_java**

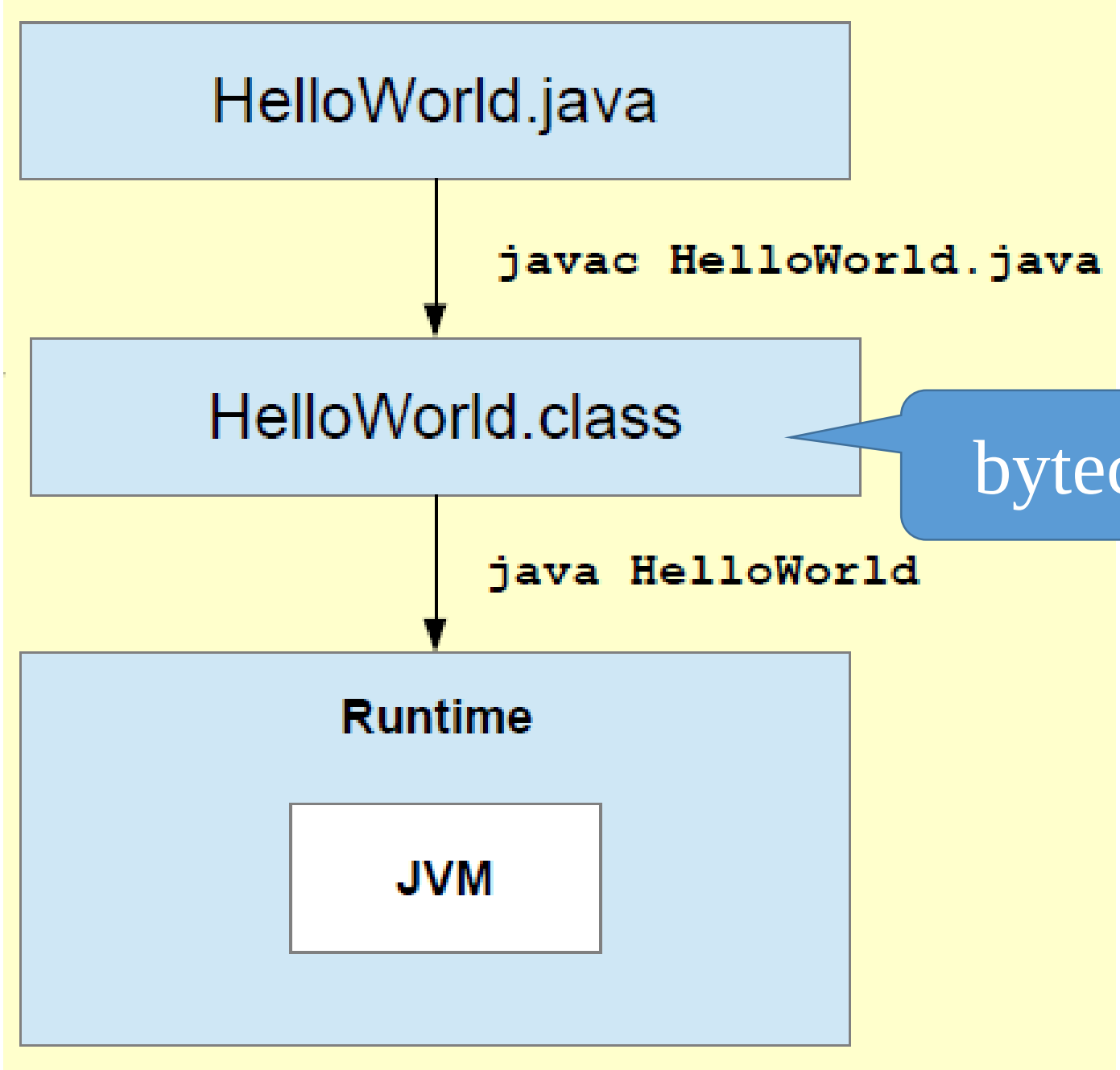
Organization of a Java Program

Packages, classes, fields, and methods

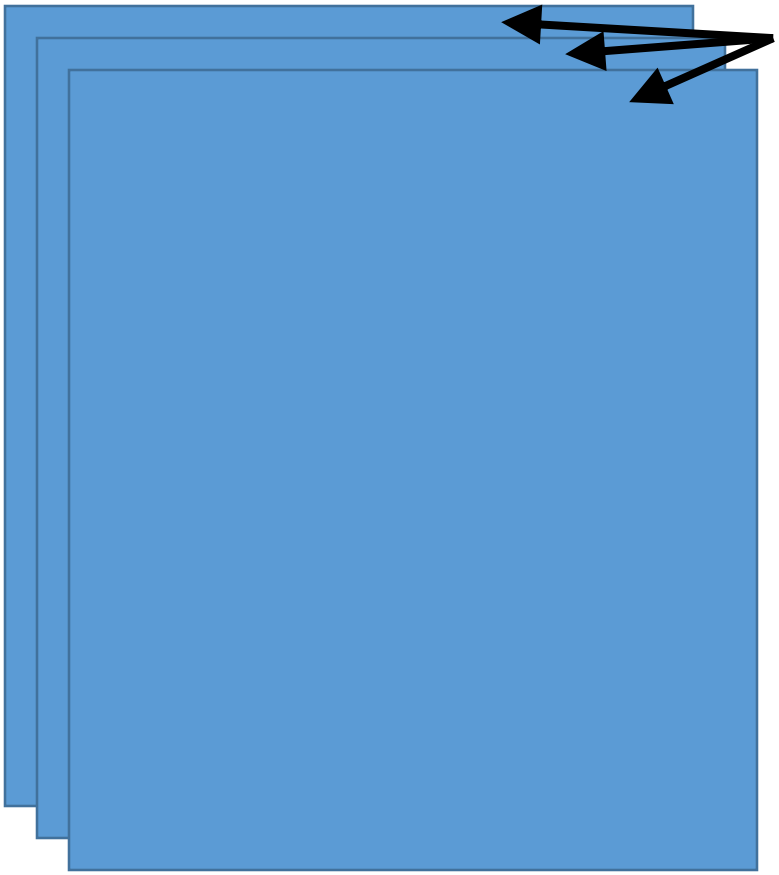
Hello World Application

Write the source code: HelloWorld.java

```
public class HelloWorld{
   public static void main( String args[] ){
      System.out.println("Hello world");
   }
}
```

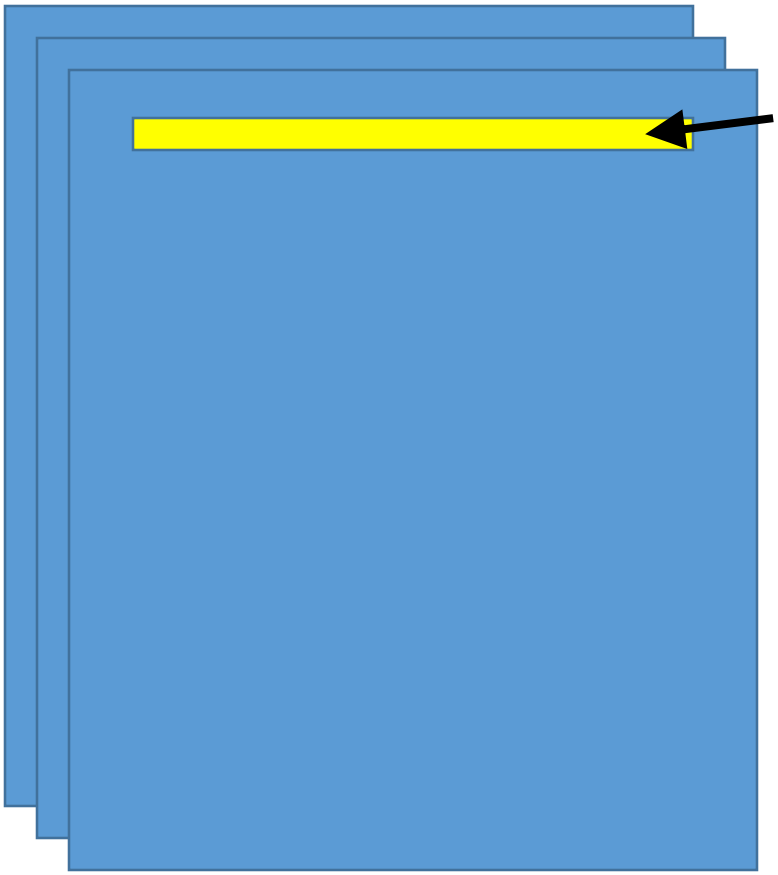

HelloWorld.java
javac HelloWorld.java
HelloWorld.class
bytecode
java HelloWorld
Runtime
JVM

Organization of a Typical Java Program



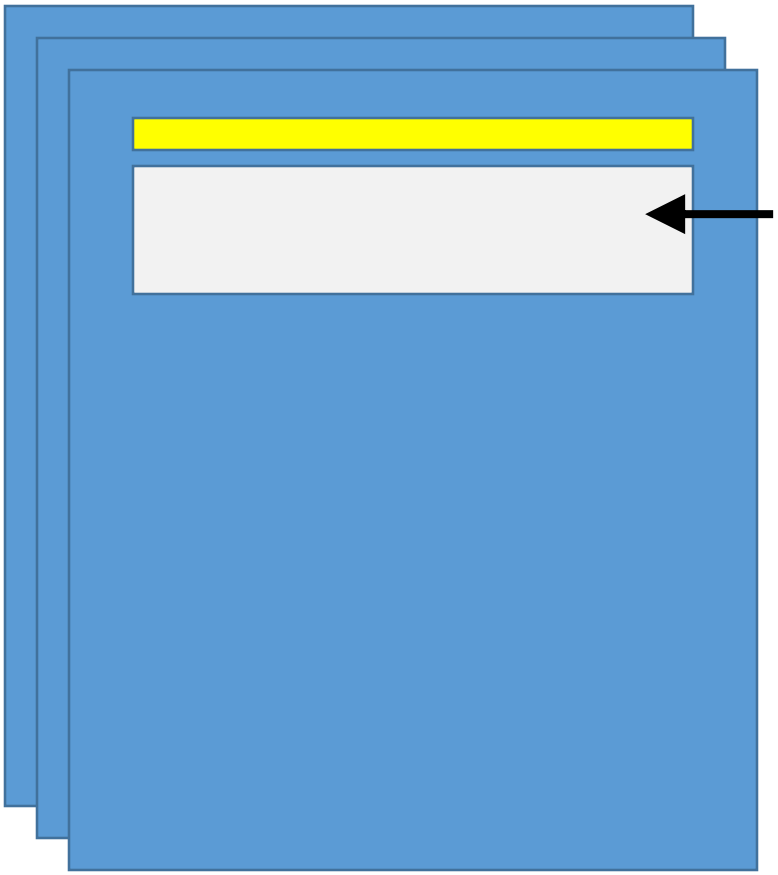
- one or more files

Organization of a Typical Java Program



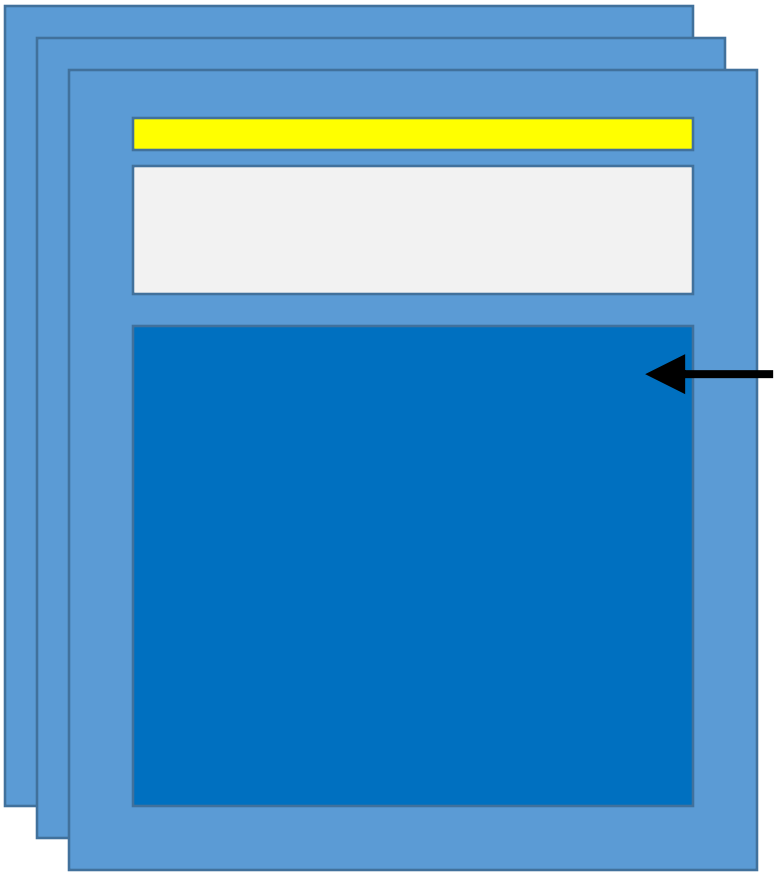
- one or more files
- zero or one package name

Organization of a Typical Java Program



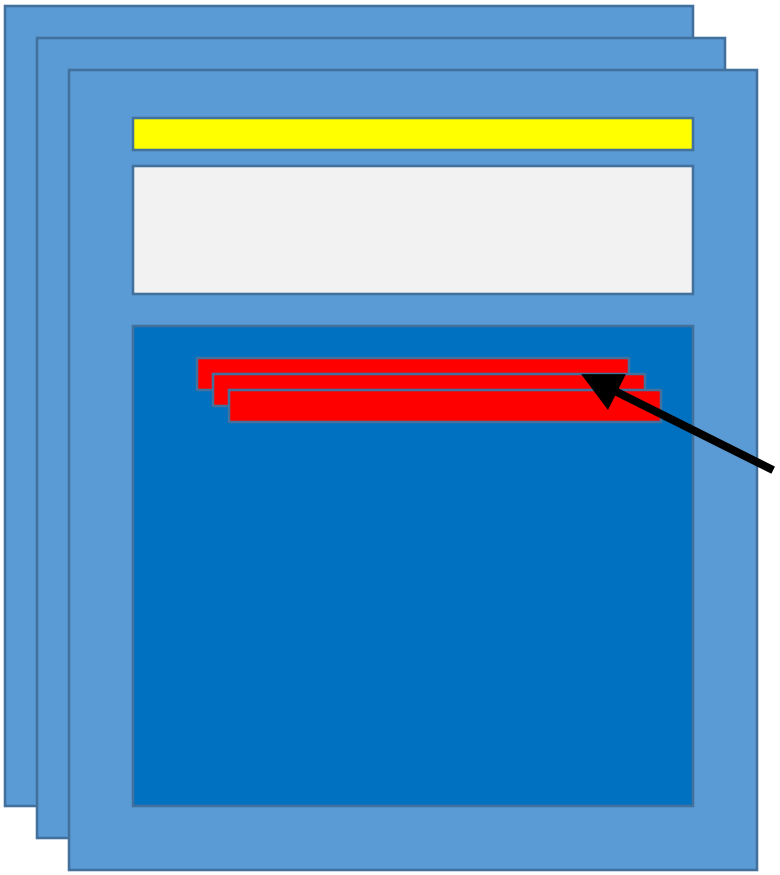
- one or more files
- zero or one package name
- zero or more import statements

Organization of a Typical Java Program



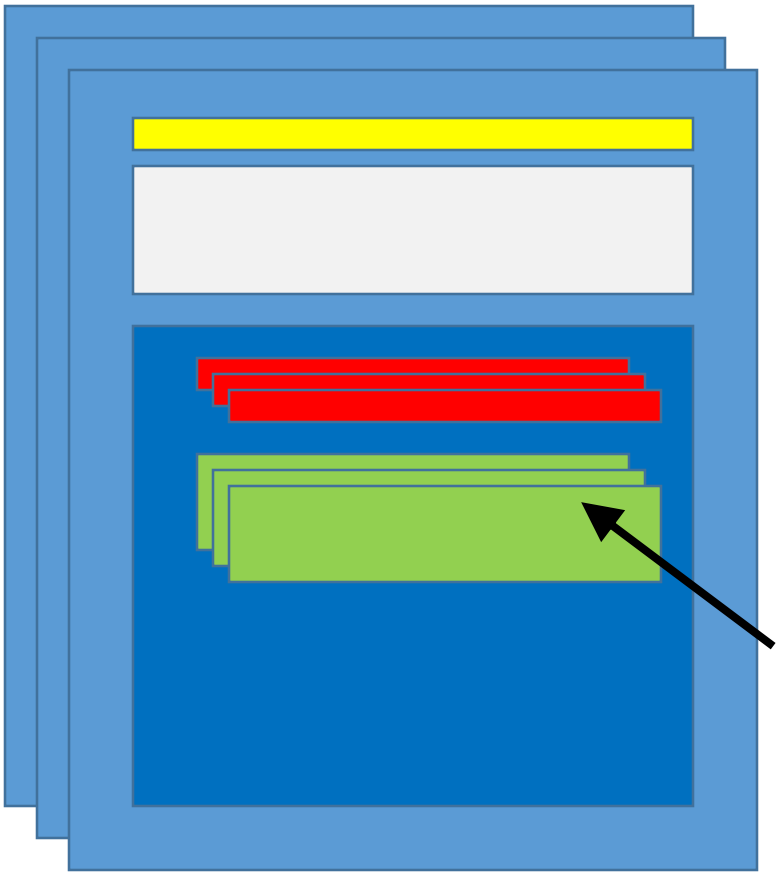
- one or more files
- zero or one package name
- zero or more import statements
- one class

Organization of a Typical Java Program



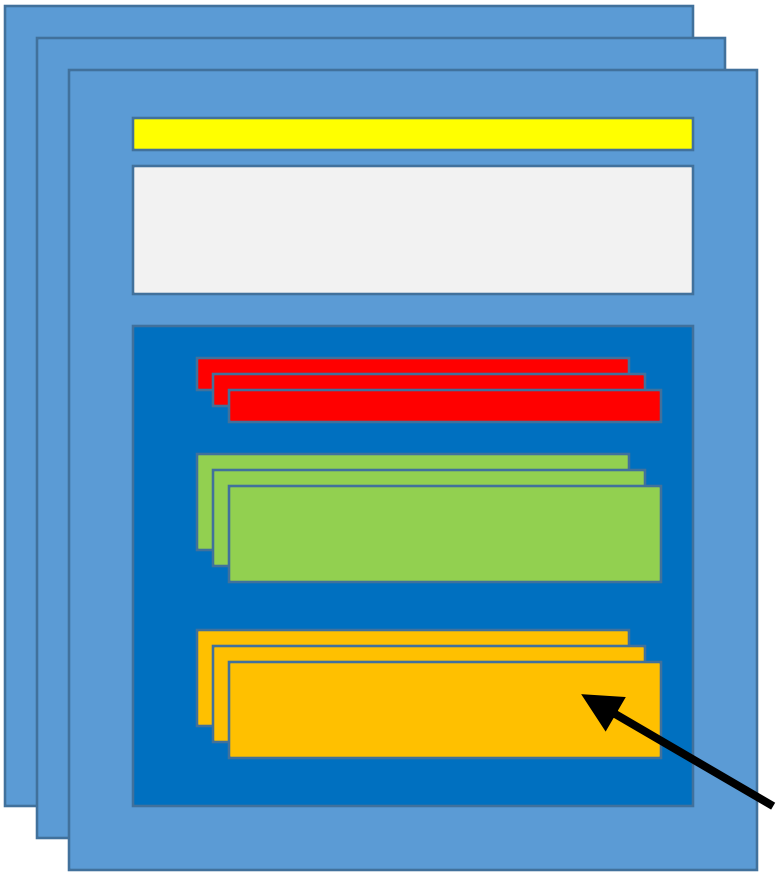
- one or more files
- zero or one package name
- zero or more import statements
- one class
- one or more fields (***class variables***)

Organization of a Typical Java Program



- one or more files
- zero or one package name
- zero or more import statements
- one class
- zero or more fields (class variables)
- zero or more more constructors

Organization of a Typical Java Program



- one or more files
- zero or one package name
- zero or more import statements
- one class
- zero or more fields (class variables)
- zero or more more constructors
- zero or more methods

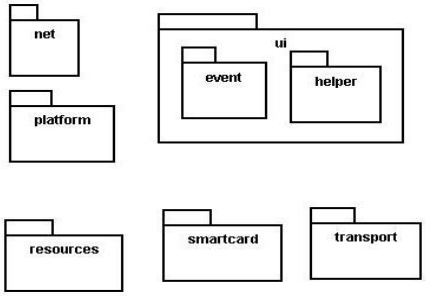
Organization of a Typical Java Program

- it's actually more complicated than this
  - static initialization blocks
  - non-static initialization blocks
  - classes inside of classes (inside of classes ...), classes inside of methods, anonymous classes
  - lambda expressions (in Java 8)

- For More details see http://docs.oracle.com/javase/tutorial/java/javaOO/index.html

Packages

- Packages are used to organize Java classes into **namespaces**
  - We organize files into different directories according to their functionality, usability as well as category they should belong to.
- A namespace is a container for names
  - the namespace also has a name



- **Hint:**
  - **Java packages can be stored in compressed files called JAR files.**

- Packages are use to organize related classes and interfaces
  - e.g., all of the Java API classes are in the package named **`java`**

package ←→ directory (folder)
class ←→ file

- Packages can contain **subpackages**
  - e.g., the package **`java`** contains packages named **`lang`**, **`util`**, **`io`**, etc.
- ***The fully qualified name of the subpackage is the fully qualified name of the parent package followed by a period followed by the subpackage name***
  - e.g., **`java.lang`**, **`java.util`**, **`java.io`**

General Overview of Java Packages API

- `javax.swing`: classes dealing with the development of GUIs.
- `java.lang`: **essential classes required by the Java language**.
- `java.text`: facilities for formatting text output.
- `java.util`: classes for storing/accessing collections of objects.
- `java.net`: for network communication.

- Packages can contain classes and interfaces
  - e.g., the package **`java.lang`** contains the classes **`Object, String, Math`**, etc.
- ***The fully qualified name of the class is the fully qualified name of the containing package followed by a period followed by the class name***
  - e.g., **`java.lang.Object, java.lang.String, java.lang.Math`**

- Packages are supposed to ensure that fully qualified names are **unique**
  - For example, if we have a class name called "**Vector**", its name would crash with the **Vector class** from JDK. However, this never happens because JDK uses **`java.util`** as a package name for the Vector class (***`java.util.Vector`***).
- This allows the compiler to disambiguate classes with the same unqualified name, e.g.,

```
your.String s = new your.String("hello");
String t = "hello";
```

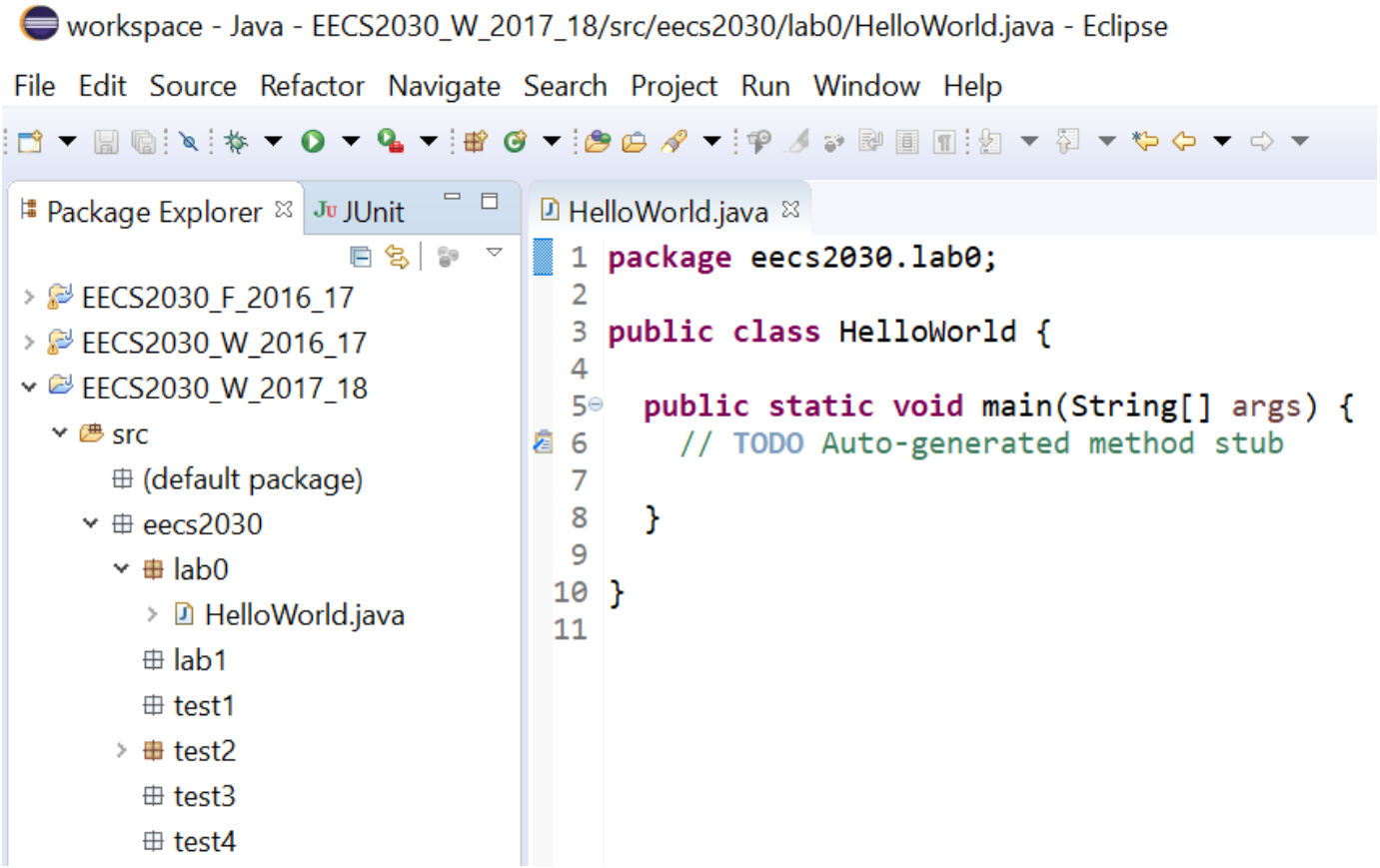
- ***How do we ensure that fully qualified names are unique?***
- By using ***package naming convention***
  - packages should be organized using your domain name in reverse, e.g.,
    - EECS domain name **`eecs.yorku.ca`**
    - package name **`ca.yorku.eecs`**

- We might consider putting everything for this course under the following package
  - **`eecs2030`**

- We might consider putting everything for this course under the following package
  - **`eecs2030`**
- Labs might be organized into subpackages:
  - **`eecs2030.lab0`**
  - **`eecs2030.lab1`** and so on
- Tests might be organized into subpackages:
  - **`eecs2030.test1`**
  - **`eecs2030.test2`** and so on

- ***Most Java implementations assume that your directory structure matches the package structure***, e.g., **package `eecs2030.lab0`**
  - there is a folder **`eecs2030`** inside the project **`src`** folder
    - there is a folder **`lab0`** inside the **`eecs2030`** folder

Eclipse – Packages overview



project folder

project sources folder

eecs2030 folder

lab0 folder

To put a class into a package, one uses the "package" statement

https://docs.oracle.com/javase/specs/jls/se10/html/jls-7.html

Importing a package

```
import packageName.*;   // all classes
import packageName.className;// one class
```

Importing a package - Static import

`import static` **packageName.className.**`*;`

Example:

```
import static java.lang.Math.*;
...
double angle = sin(PI / 2) + ln(E * E);
```

- Static import allows you to refer to the ***members of another class without writing that class's name***.
- Should be used rarely and only with classes whose contents are entirely static "***utility***" code.

Package access

- Java provides the following access modifiers:
  - **`public`** : Visible to all other classes.
  - **`private`** : Visible only to the current class (and any nested types).
  - **`protected`** : Visible to the current class, any of its subclasses, and any other types within the same package.
  - **default (package): Visible to the current class and any other types within the same package.**

Notes on the import statement

- Import ONLY imports public classes from the specified package
  - `Classes which are not public cannot be referenced from outside their package.`
- There is no way to "***import all classes except one***"
  - `import either imports a single class or all classes within the package`
  - `Note: importing has no runtime or performance implications. It is only importing a namespace so that the compiler can resolve class names.`
- **Import statements must appear at the top of the file after the package statement and before any class or interface definitions.**

Objects in JAVA

Basics

Define Your Own Objects

- In the previous course, you may have already gained experience in **defining your own data structures** (a.k.a. ***data types, objects***) that you used within your program in order to group various data elements together.
  - We create this object by defining a ***class***.
  - Each class that we define represents a new ***type*** (or ***category***) of object.

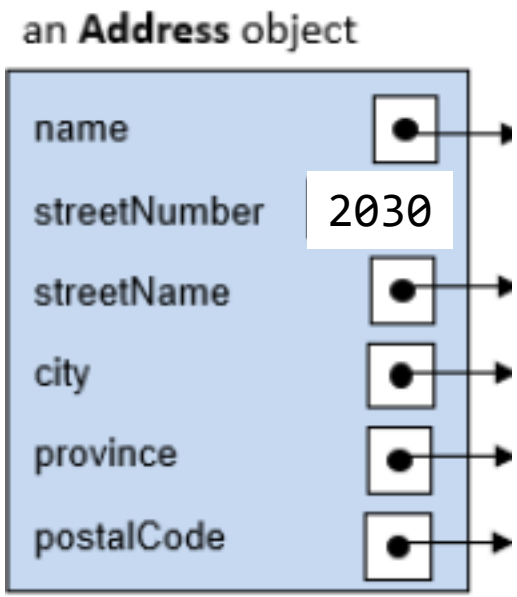
```
class Address {
      String      name;
      int         streetNumber;
      String      streetName;
      String      city;
      String      province;
      String      postalCode;
}
```

Objects

- *A **object** **represents multiple pieces of information that are grouped together.***
- A primitive data type (e.g., **int**, **float**, **char**) represents a **single** simple piece of information.
- An ***object***, however, is a **bundle** of data, which can be made up of multiple primitives or possibly other objects as well.

Objects

- Once we define Address class/object, then we were allowed to create **Address objects** and use them within our programs.




```
Address addr;
addr = new Address();
addr.name = "EECS Student One";
addr.streetNumber = 2030;
addr.streetName = "EECS Department ST";
addr.city = "Tornoro";
addr.province = "ON";
addr.postalCode = "M3J 1P3";
System.out.print(addr.name + " lives at ");
System.out.println(addr.streetNumber + " " + addr.streetName);
```

Overview

- An object can contain variables as well as methods.
- Variables and methods are called **members** of class.

Note: ***Variable in an object is called a field, data, attributes or instance variables.***

- Generally, fields are defined as private so they can't be seen from outside the class.
- May add getter methods (functions) and setter methods (procedures) to allow access to some or all fields.
- *We use constructors, to initialize fields of a new object during evaluation of a new-expression.*

Example

```
public class Circle {
 public double x, y;  // centre coordinate
 public double r;     //  radius of the circle

}
```

The fields (data) are also called the *instance* ***variables***.

```
public class Circle {

      public double x, y; // centre of the circle
      public double r;    // radius of circle

      //Methods to return circumference and area
      public double circumference() {
            return 2*3.14*r;
      }
      public double area() {
            return 3.14 * r * r;
     }
}
```

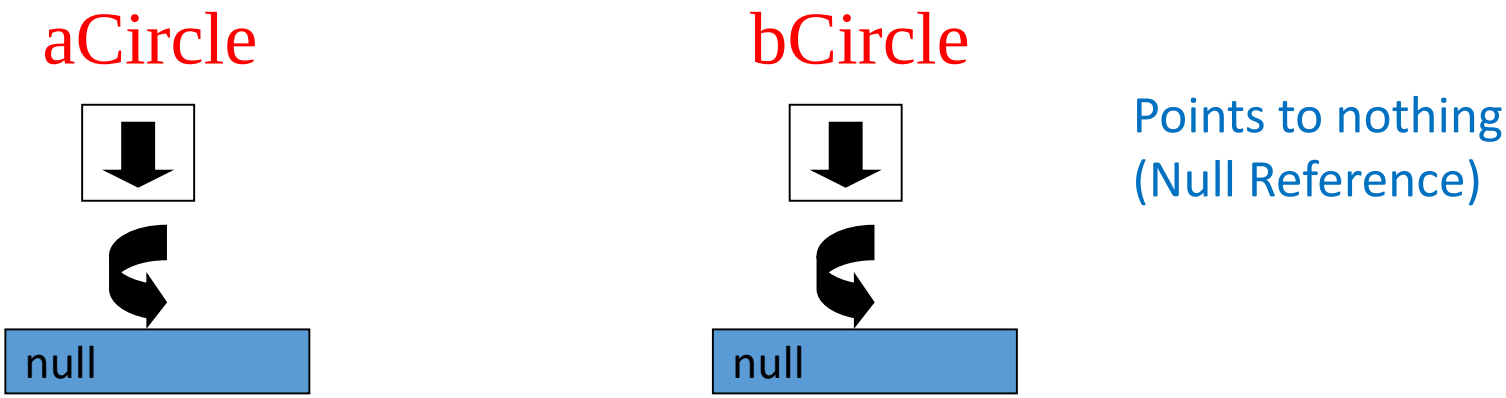
Method Body

Data Abstraction

- Declare the Circle class, have created a new data type – **Data Abstraction**
- Can define variables (objects) of that type:

```
Circle  aCircle;
Circle  bCircle;
```

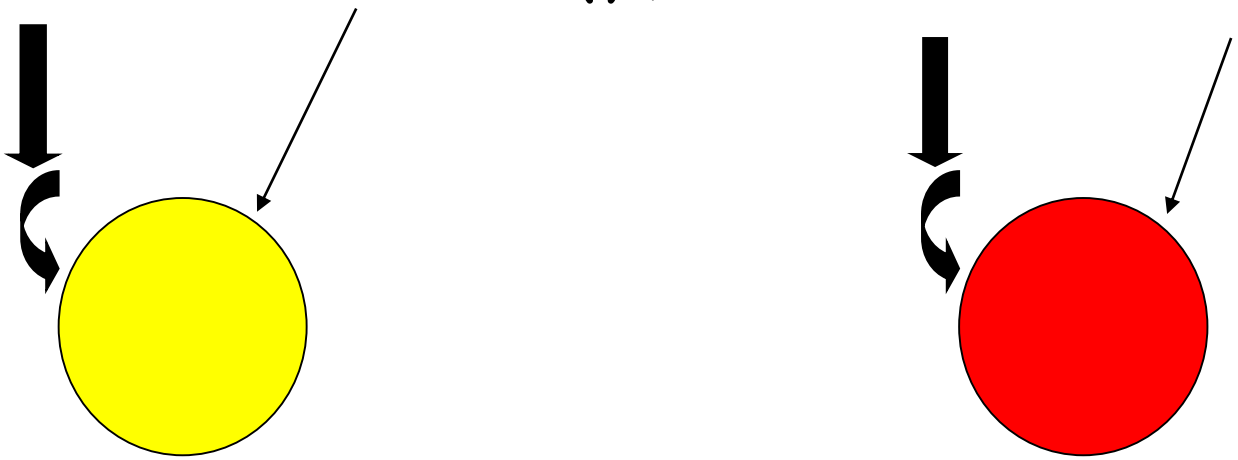
- **`aCircle, bCircle`** simply refers to a Circle object, not an object itself.



Creating objects of a class

- Objects are created dynamically using the *new* keyword.

```
aCircle = new Circle(); bCircle = new Circle();
```



References

- https://docs.oracle.com/javase/10/docs/api/overview-summary.html
- https://www.eecs.yorku.ca/course_archive/ [look for EECS 2030]