Introduction to Computer Science II

CSE1030Z

Academic Support Programs: Bethune

‣ having trouble with your FSC and LSE courses?

  ‣ consider using the Academic Support Programs at Bethune College

  ‣ PASS

    ‣ free, informal, structured, facilitated study groups: http://bethune.yorku.ca/pass/

  ‣ peer tutoring

    ‣ free, one-on-one, drop-in tutoring: http://bethune.yorku.ca/tutoring/

Academic Support Programs: Bethune

‣ Bethune College is looking for Class Representatives

   ‣ **http://bethune.yorku.ca/classreps/**

ACM Programming Contest

- http://icpc.baylor.edu

- EECS team website
  - https://wiki.cse.yorku.ca/project/ACM

- first contest
  - Fri Sep 13 at 13:13 in Lassonde 1004

Who Am I?

- Dr. Burton Ma
- office
  - Lassonde 2046
  - hours : see syllabus on course web page
- email
  - `burton@cse.yorku.ca`

Course Format

‣ everything you need to know is on the course website

    ‣ **`http://www.eecs.yorku.ca/course/1030`**

‣ labs start next Tuesday (Sep 17)

CSE1030 Overview

- in CSE1020, you learned how to use objects to write Java programs
  - a Java program is made up of one or more interacting objects
  - each object is an instance of a class

- where do the classes come from?

- in CSE1030, you will learn how to design and implement classes
  - introduction to concepts in software engineering and computer science

What You Should Know from CSE1020

‣ how to read an API

   ‣ determine what package a class is located in

   ‣ determine what the class/interface/field/method is supposed to do

   ‣ determine the name of a method

   ‣ determine what types a method requires for its parameters

   ‣ determine what type a method returns

   ‣ determine what exceptions might be thrown

What You Should Know from CSE1020

▸ create and use primitive type variables and their associated operators

  ▸ `int`, `double`, `boolean`, `char`

What You Should Know from CSE1020

- create (using a constructor) and use reference variables
  - e.g., `type.lib.Fraction`, `java.util.Date`
  - `Random`, `String`, `List`, `Set`, `Map`

What You Should Know from CSE1020

‣ understand the difference between primitive and reference types

  ‣ memory diagrams

What You Should Know from CSE1020

‣ understand the difference between **==** and **equals**

What You Should Know from CSE1020

▸ use class methods (and fields)

  ▸ e.g.,

```
double value = Math.sqrt(2.0);
```

▸ use instance methods (and fields)

  ▸ e.g.,

```
String s = "hello";
String t = s.toUpperCase();
```

What You Should Know from CSE1020

- **if** statements

- e.g.

```
if (grade >= 65) {
    System.out.println("Go to second year");
}
else {
    System.out.println("Try again");
}
```

What You Should Know from CSE1020

‣ **for** loops

‣ e.g., for some **String** reference **s**

```
for (int i = 0; i < s.length(); i++) {
  char c = s.charAt(i);
  if (c == 'a') {
    System.out.println(s + " contains an \'a\'");
    break;
  }
}
```

What You Should Know from CSE1020

▸ **for** each loops

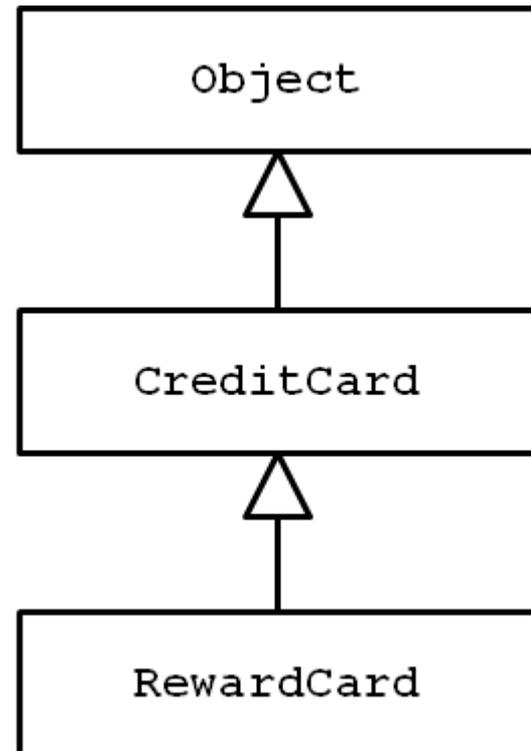▸ e.g., for some **List<String>** reference **t**

```
for (String s : t) {
  for (int i = 0; i < s.length(); i++) {
    char c = s.charAt(i);
    if (c == 'a') {
      System.out.println(s + " contains an \'a\'");
      break;
    }
  }
}
```

What You Should Know from CSE1020

▸ the difference between aggregation and composition
▸ the differences between aliasing, shallow copying, and deep copying

What You Should Know from CSE1020

▸ inheritance and substitutability

```
┌──────────────────────┐
│       Object         │
└──────────────────────┘
           △
           │
┌──────────────────────┐
│     CreditCard       │
└──────────────────────┘
           △
           │
┌──────────────────────┐
│     RewardCard       │
└──────────────────────┘
```

What You Should Know from CSE1020

- what an exception is
- the difference between a checked and unchecked exception
- how to handle exceptions (`try` and `catch`)

What You Should Know from CSE1020

▸ style

```
public class hairsOnHead
{
  public static void main(String[] args) {
     int Diameter = 17;
     double f = 0.5;
     double areaCovered=f*Math.PI*Diameter*Diameter;
     int d = 200;
     double numberofhairs = areaCovered * d;
     System.out.print("The number of hairs on a human head is ");
  System.out.println(numberofhairs);
  }
}
```

class names should start with a capital letter

inconsistent brace alignment

variable names should start with a lowercase letter; magic number

variable names should be informative; magic number

1 space around operators

variable names should be informative; magic number

variable names should use camelcase

inconsistent indenting

Organization of a Java Program

Packages, classes, fields, and methods

In This Lecture

1. demonstrate the use of eclipse by solving a CSE1020 eCheck problem

2. review the organization of a typical CSE1020 Java program

3. improve the organization of the program by writing a method

4. explain the organization of a typical Java program that uses packages and multiple classes

eCheck04A

‣ in a nutshell:

    ‣ write a program that computes the fraction

$$A = \frac{x + y}{z + t}$$

    where x, y, z, and t are proper fractions entered by a user from the command line

eCheck04A Sample Output

```
For each fraction enter its numerator/denominator,

pressing ENTER after each

Enter x

83

100

Enter y

5

9

Enter z

667

1000

Enter t

-2

3

A = 12470/3 = 4156 2/3 = 4156.666666666667
```
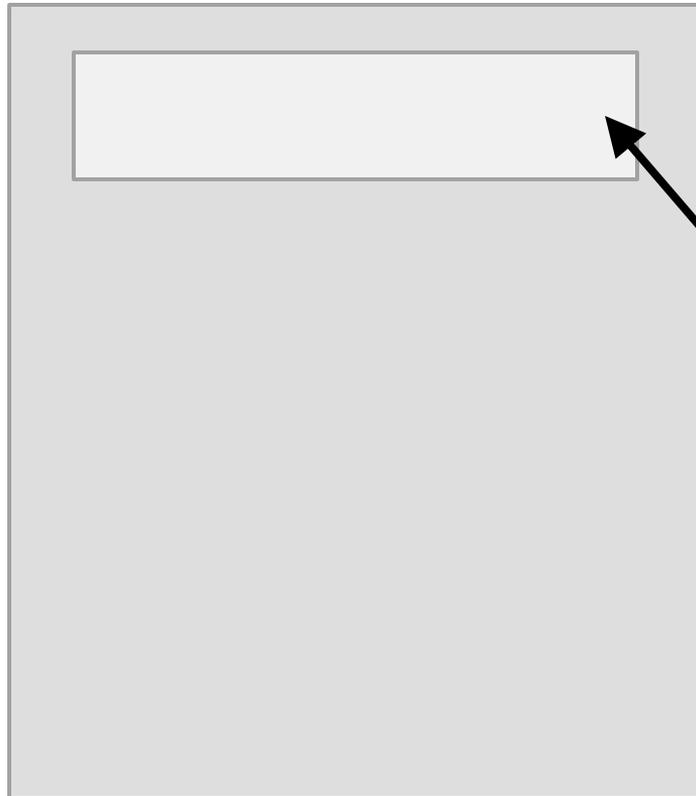
eclipse Demo Here

‣ if you missed this class then you missed this demo
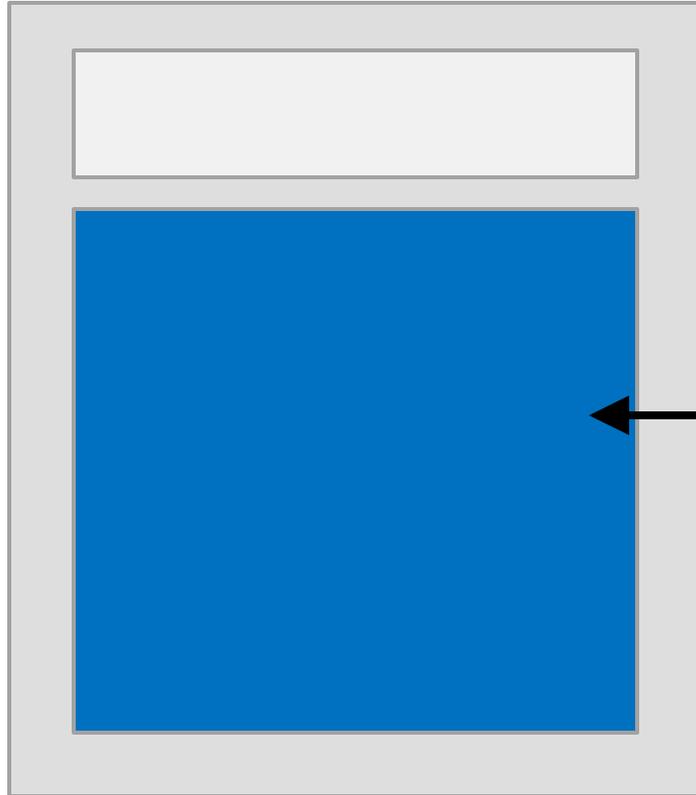
Organization of a CSE1020 Program

▸ one file
  ▸ `Check04A.java`

Organization of a CSE1020 Program
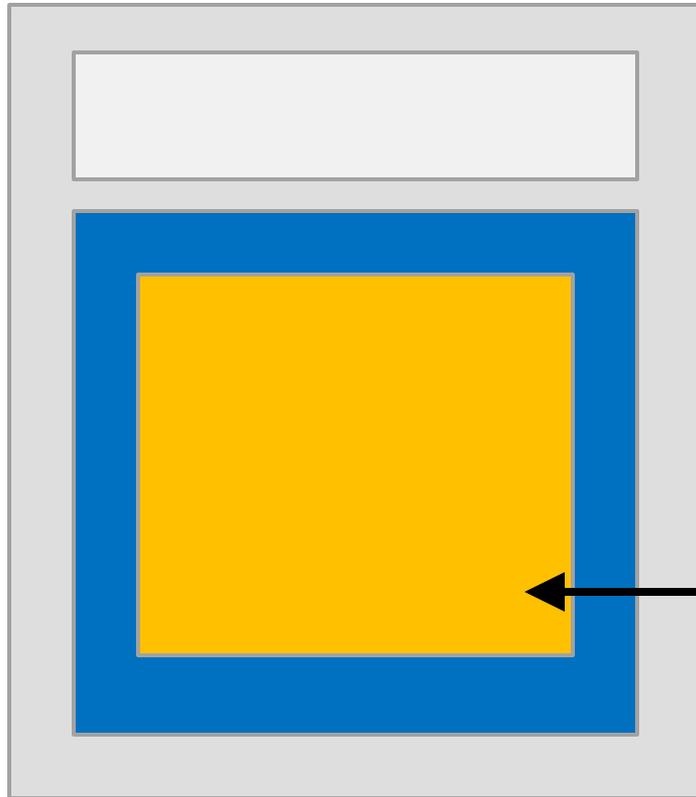
▸ one file
  ▸ `Check04A.java`

▸ zero or more import statements

Organization of a CSE1020 Program

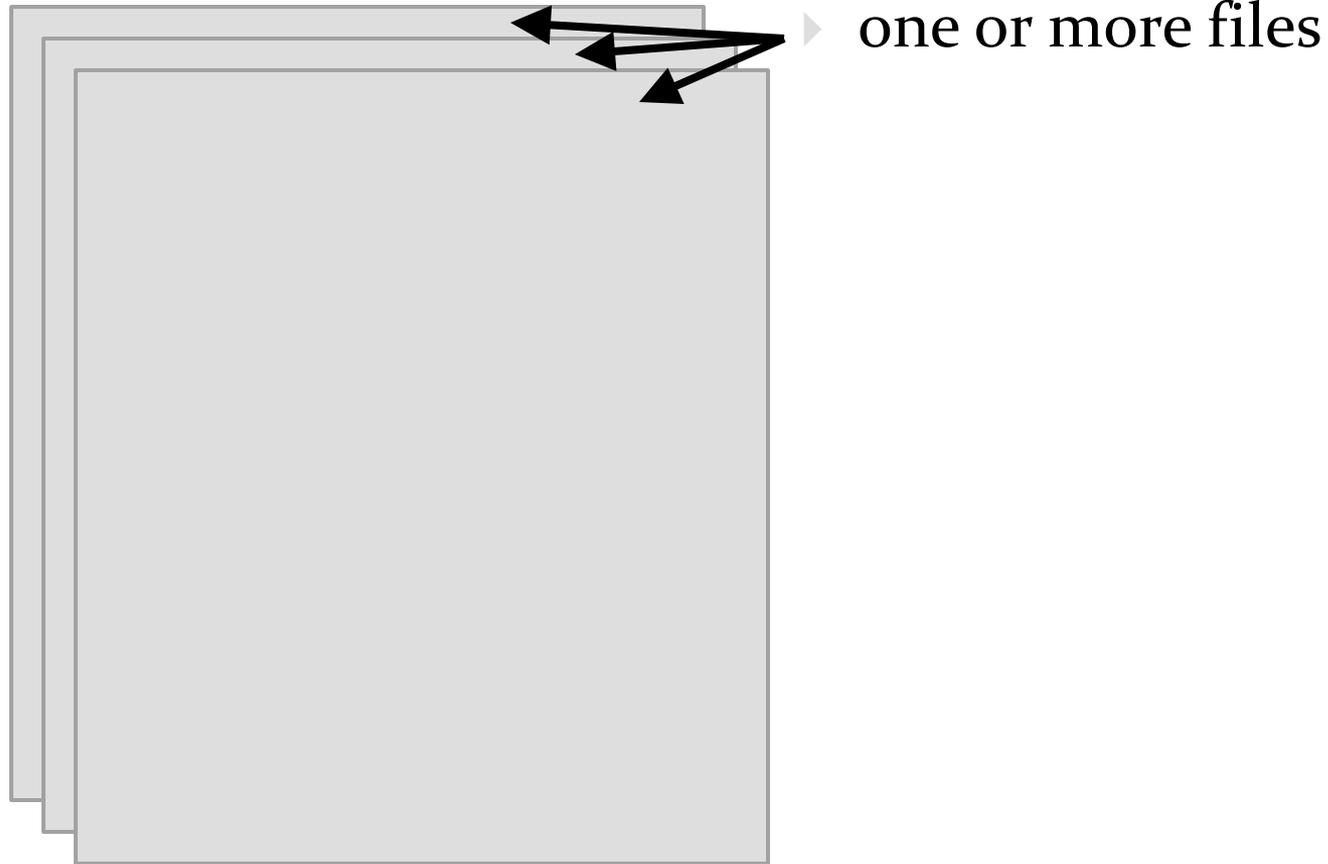▸ file
  ▸ `Check04A.java`
▸ zero or more import statements
▸ one class
  ▸ `Check04A`

Organization of a CSE1020 Program
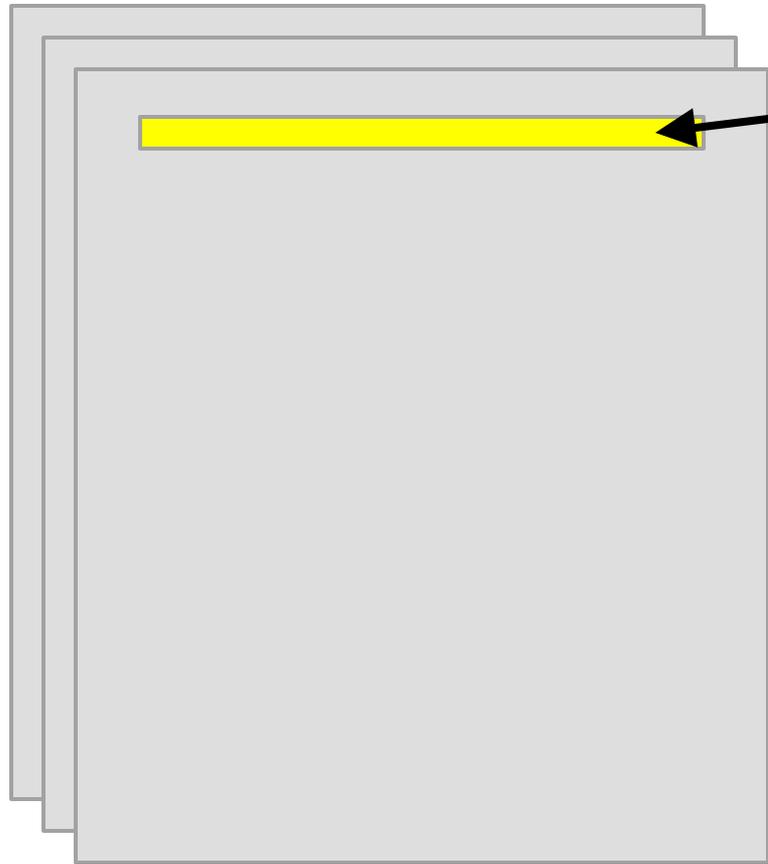

- file
  - Check04A.java
- zero or more import statements
- one class
  - **Check04A**
- one static method
  - **main**

Organization of a Typical Java Program

one or more files

Organization of a Typical Java Program

▸ one or more files

▸ one package name

Organization of a Typical Java Program
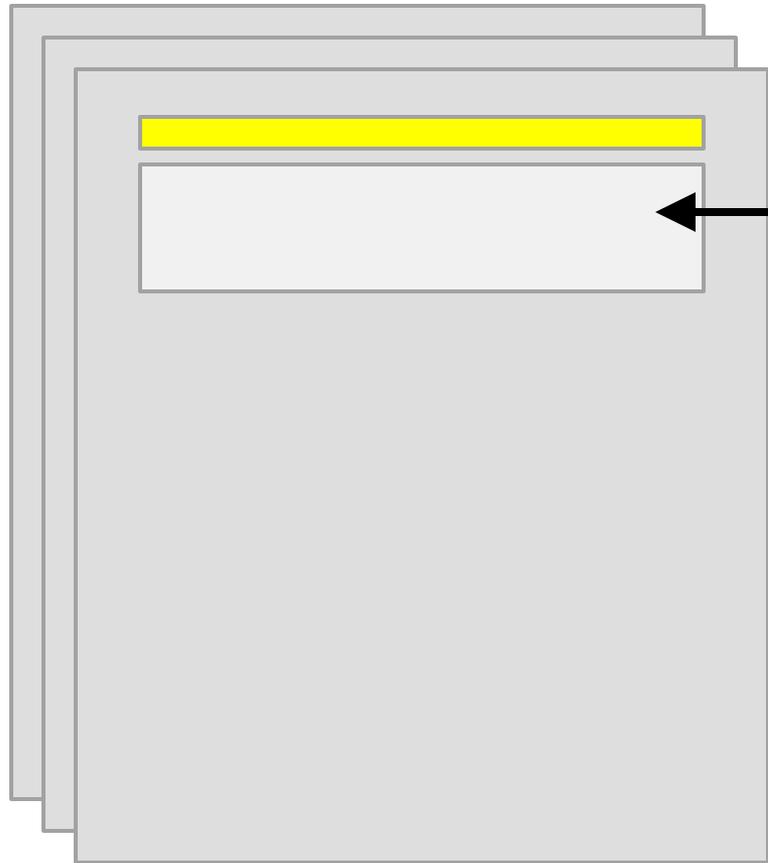
▸ one or more files

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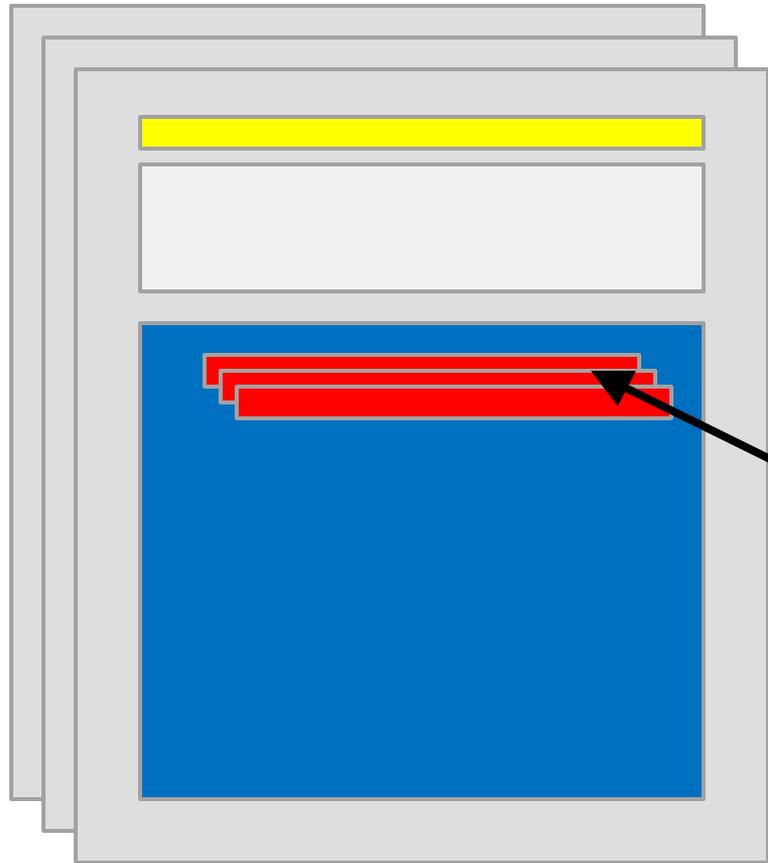
- one or more files
- zero or one package name
- zero or more import statements
- one class
- one or more fields (class variables)

Organization of a Typical Java Program
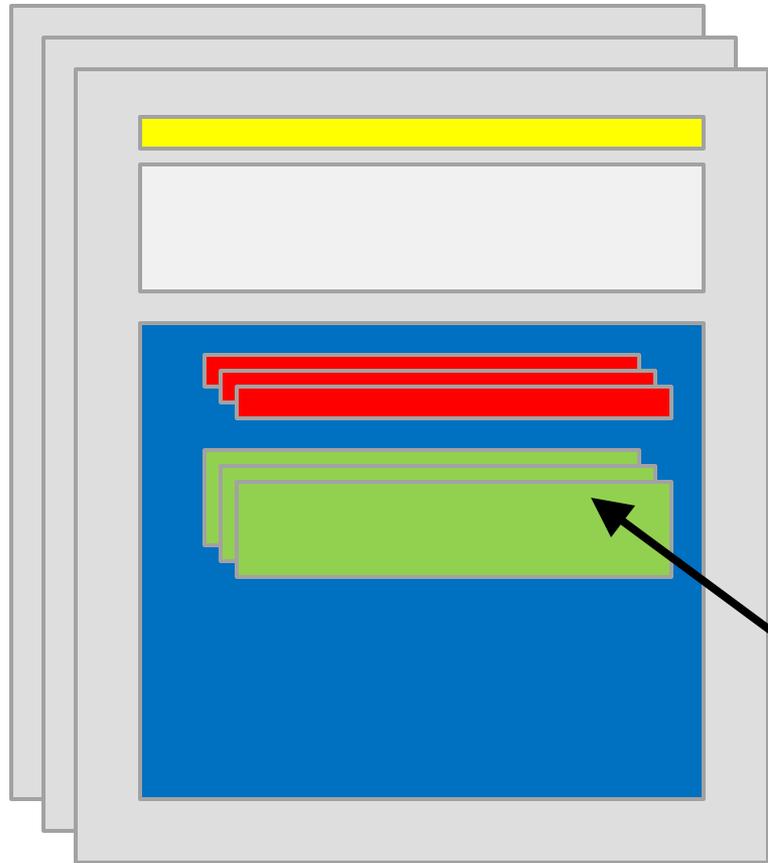
- one or more files
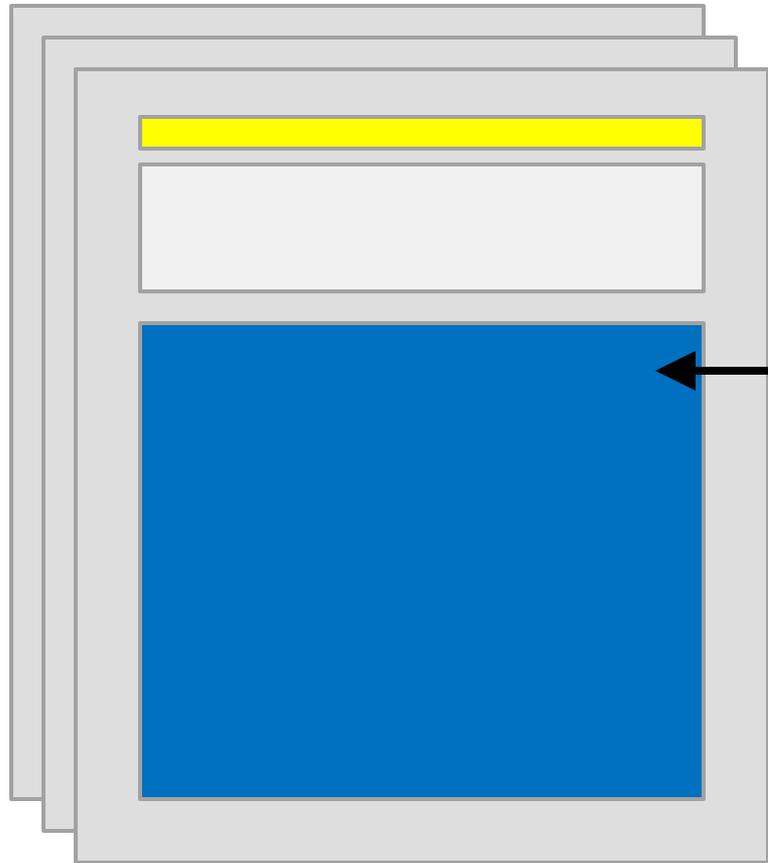- zero or one package name
- zero or more import statements
- one class
- zero or more fields (class variables)
- zero or more more constructors

Organization of a Typical Java Program
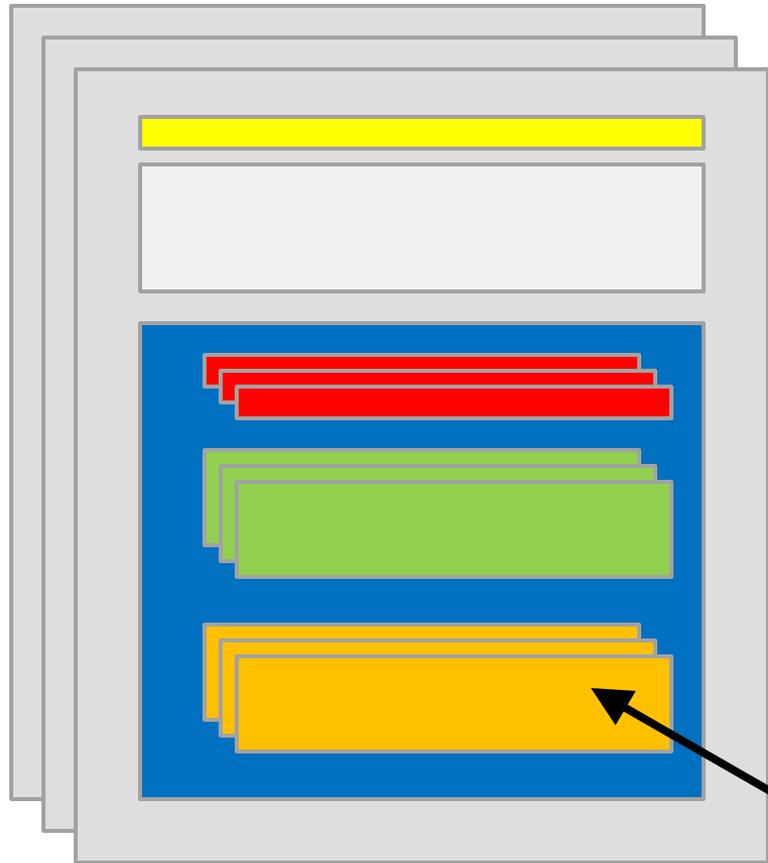
▸ one or more files

▸ 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

‣ see http://docs.oracle.com/javase/tutorial/java/javaOO/index.html

Packages

▸ packages are used to organize Java classes into namespaces

▸ a namespace is a container for names

   ▸ the namespace also has a name

Packages

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

Packages

▸ 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`**

Packages

- 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

- packages are supposed to ensure that fully qualified names are unique

- this allows the compiler to disambiguate classes with the same unqualified name, e.g.,

```
your.Fraction f = new your.Fraction(1, 3);
type.lib.Fraction g = new type.lib.Fraction(1, 3);
```

Packages

- how do we ensure that fully qualified names are unique?

- 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
  - `ca.yorku.eecs.cse1030`

Packages

▸ most Java implementations assume that your directory structure matches the package structure, e.g.,

▸ there is a sequence of folders `ca\yorku\eecs\cse1030` inside the project `src` folder

eclipse workspace folder	◢ 📁 1030F13_14
	▷ 📁 .metadata
project folder	◢ 📁 eCheck04A
	📁 .settings
	▷ 📁 bin
project sources folder	◢ 📁 src
	◢ 📁 ca
	◢ 📁 yorku
	◢ 📁 eecs
Java source files	📁 cse1030
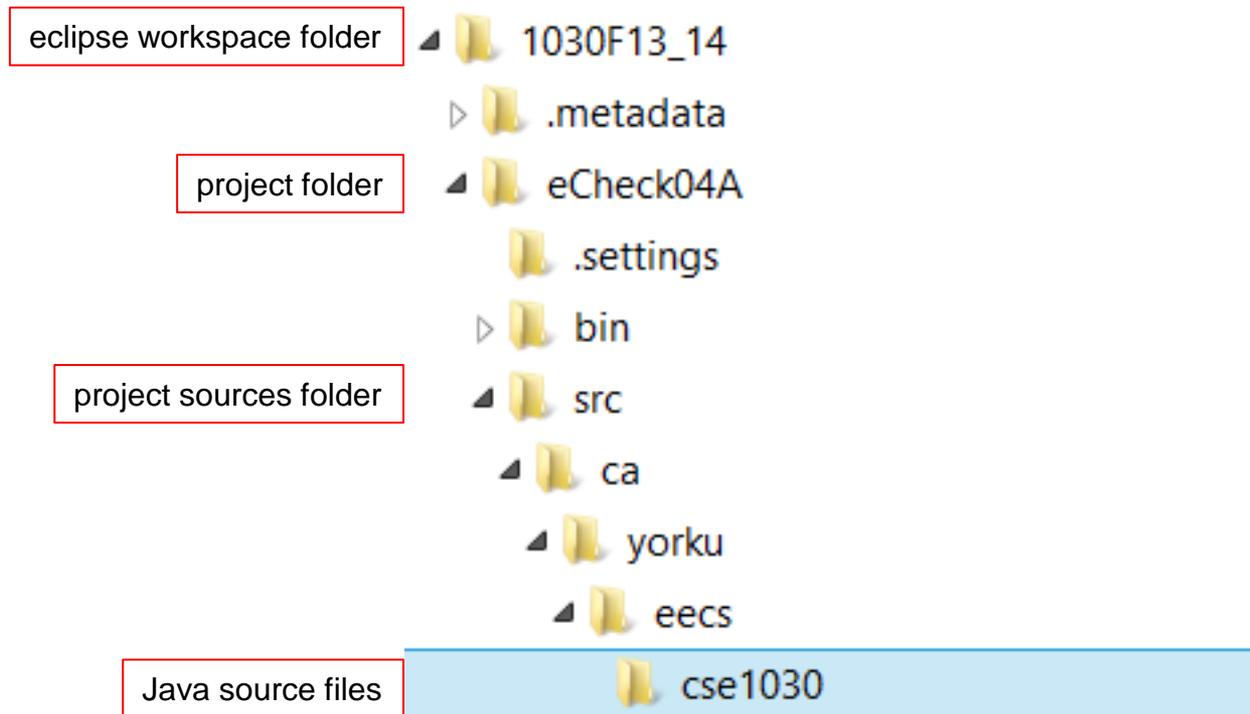
Things For You to do this Week

‣ get a CSE account if you do not already have one

‣ do Lab oo to get (re)acquainted with eclipse and the CSE labs

  ‣ available tomorrow

‣ review CSE1020

CSE1020 Review Questions

CSE1020 Review

▸ what does the following program print?

```
public class Puzzle01
{
  public static void main(String[] args)
  {
      System.out.print("C" + "S" + "E");
      System.out.println('1' + '0' + '3' + '0' + 'z');
  }
}
```

CSE1020 Review

▸ which of the following methods are associated with a class?

```
static boolean disjoint(Collection<?> c1, Collection<?> c2)
```

```
void setIcon(Icon newIcon)
```

```
String toString()
```

```
static int round(double a)
```

```
static void showMessageDialog(Component parent, Object message)
```

CSE1020 Review

‣ what is the return type for each of the following methods?

```
static boolean disjoint(Collection<?> c1, Collection<?> c2)
```

```
void setIcon(Icon newIcon)
```

```
String toString()
```

```
static int round(double a)
```

```
static void showMessageDialog(Component parent, Object message)
```

CSE1020 Review

▸ how many parameters do each of the following methods have, and what are their types?

```
static boolean disjoint(Collection<?> c1, Collection<?> c2)
```

```
void setIcon(Icon newIcon)
```

```
String toString()
```

```
static int round(double a)
```

```
PrintStream printf(String format, Object... args)
```

CSE1020 Review

‣ what is a method precondition

‣ what is a method postcondition?

‣ what happens if a precondition is violated?

‣ who is responsible if a postcondition is false?

CSE1020 Review

- a `type.lib.Fraction` object has two attributes: a numerator and a denominator
- draw the memory diagram for the following program
  - after line 1 completes
  - after line 2 completes

```
import type.lib.Fraction;

public class Fraction1 {
  public static void main(String[] args) {
    Fraction f = new Fraction(1, 2);    // 1
    f.add(new Fraction(3, 4));          // 2
  }
}
```

CSE1020 Review

▸ class **X** is an aggregation of one **Y**; it has a method **getY** that returns a reference to its **Y** object

▸ what are the values of **sameState** and **sameObject**?

```
Y y = new Y();
X x = new X(y);     // x has a reference to y
boolean sameState = y.equals(x.getY());
boolean sameObject = y == x.getY();
```

CSE1020 Review

‣ class **X** is an composition of one **Y**; it has a method **getY** that returns a reference to its **Y** object

‣ what are the likely values of **sameState** and **sameObject**?

```
Y y = new Y();
X x = new X(y);    // x uses composition with y
boolean sameState = y.equals(x.getY());
boolean sameObject = y == x.getY();
```

CSE1020 Review

▸ class **X** is an composition of one **Y**; it has a method **getY** that returns a reference to its **Y** object

　▸ furthermore, **Y** is immutable

▸ what are the likely values of **sameState** and **sameObject**?

```
Y y = new Y();
X x = new X(y);    // x uses composition with y
boolean sameState = y.equals(x.getY());
boolean sameObject = y == x.getY();
```

CSE1020 Review

▸ consider the following UML diagram

```
┌─────────────────────────┐
│        Object           │
└─────────────────────────┘
            △
            │
┌─────────────────────────┐
│       CreditCard        │
└─────────────────────────┘
            △
            │
┌─────────────────────────┐
│       RewardCard        │
└─────────────────────────┘
```
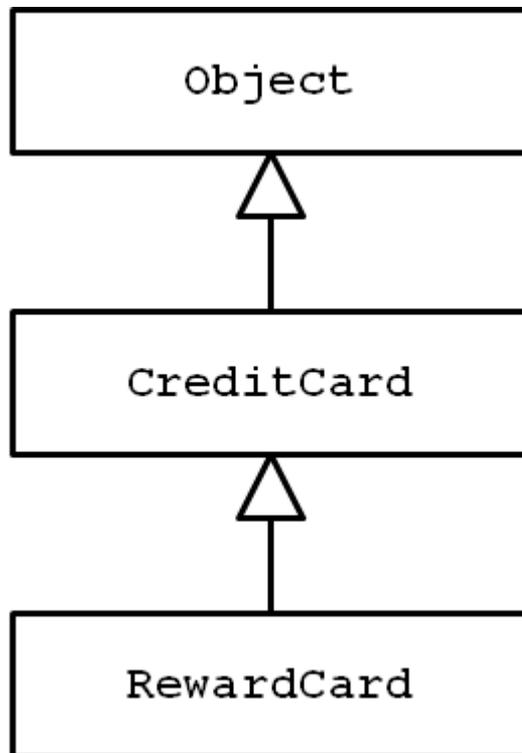
▸ which statements are true?

1. **Object** is a **CreditCard**
2. **CreditCard** is an **Object**
3. **RewardCard** is an **Object**
4. **RewardCard** is a **CreditCard**
5. a **CreditCard** is usable anywhere a **RewardCard** is required
6. a **RewardCard** is usable anywhere a **CreditCard** is required

CSE1020 Review

- **`t`** is a reference to a **`List<String>`** object
- write some code that prints out each element of **`t`**

CSE1020 Review

▸ **p** is a reference to a **Map<String, Integer>** object
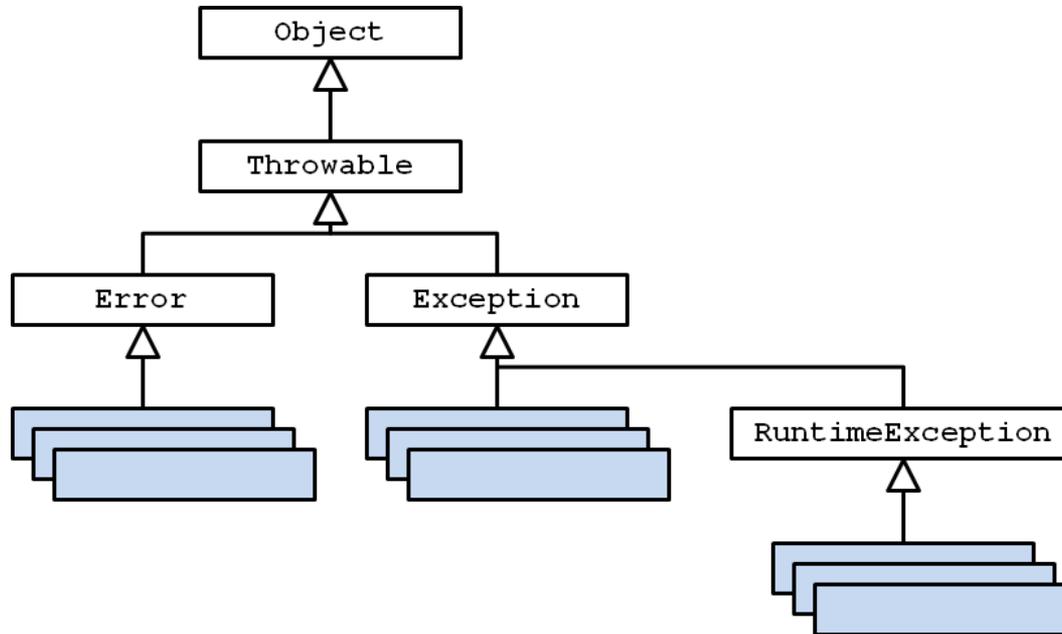▸ write some code that prints out each key-value pair of **p**

CSE1020 Review

‣ consider the UML diagram for Java exceptions:

```
                    ┌──────────┐
                    │  Object  │
                    └──────────┘
                         △
                    ┌──────────┐
                    │ Throwable│
                    └──────────┘
                         △
         ┌───────────────┴───────────────┐
    ┌──────────┐                    ┌──────────┐
    │  Error   │                    │Exception │
    └──────────┘                    └──────────┘
         △                               △
                                         └──────────┐
    [boxes]              [boxes]        ┌────────────────────┐
                                        │  RuntimeException   │
                                        └────────────────────┘
                                                 △
                                            [boxes]
```
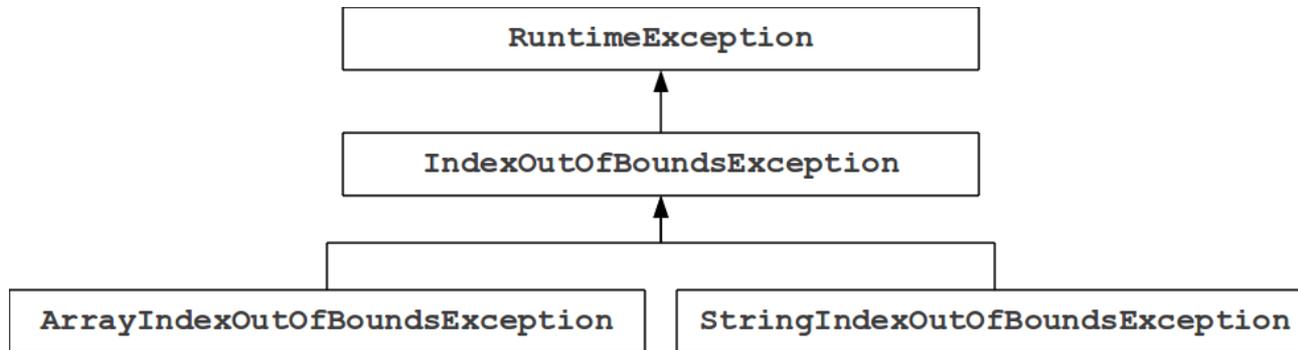
‣ checked exceptions are subclasses of ... ?
‣ unchecked exceptions are subclasses of ... ?

CSE1020 Review

▸ consider the UML diagram for some common exceptions:

```
┌──────────────────────────────────┐
│         RuntimeException          │
└──────────────────────────────────┘
                  ▲
┌──────────────────────────────────┐
│     IndexOutOfBoundsException     │
└──────────────────────────────────┘
                  ▲
        ┌─────────┴─────────────────────────┐
┌───────────────────────────────┐ ┌──────────────────────────────────┐
│ ArrayIndexOutOfBoundsException │ │ StringIndexOutOfBoundsException  │
└───────────────────────────────┘ └──────────────────────────────────┘
```

▸ will the following code fragment compile?

```
try { // some legal code not shown here }
catch (IndexOutOfBoundsException e) { // not shown }
catch (StringIndexOutOfBoundsException e) { // not shown }
```

CSE1020 Review

▸ more questions can be found here:

 ▸ http://www.eecs.yorku.ca/course_archive/2011-12/F/1020/practice.shtml