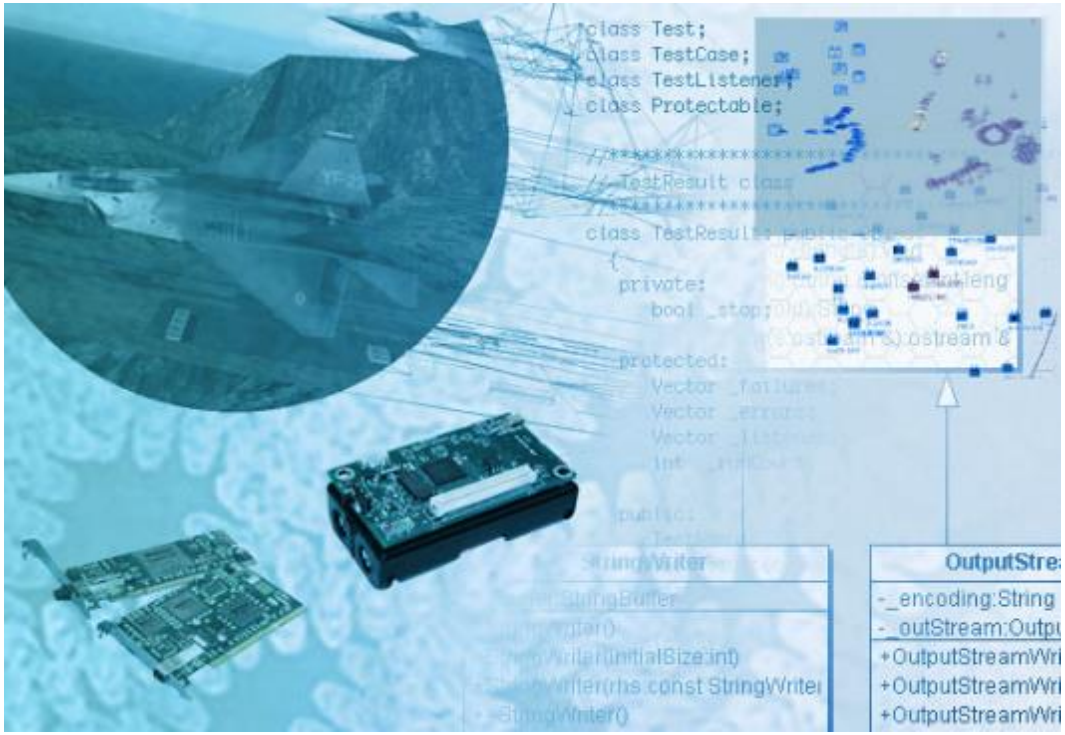
CSYE 6200

CONCEPTS OF OBJECT-ORIENTED DESIGN
SESSION 5

MARK G. MUNSON



THE LECTURE

- **Recap**
- **Diagraming with UML**
- **Packages – Organizing your code**
- **Errors and Error Handling**
  - Exceptions
  - Throw & Throws
  - Try-Catch
- **FileWriter Introduction**

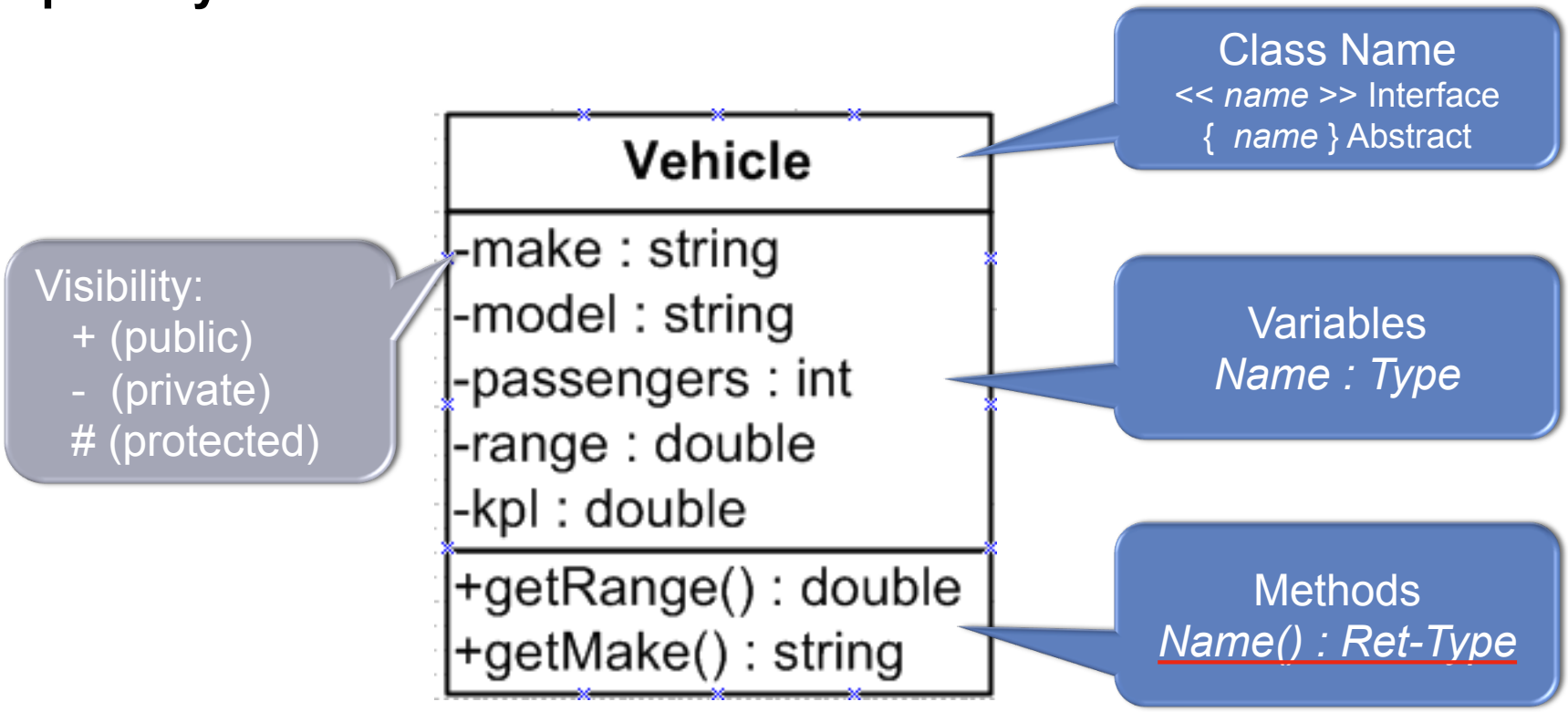
RECAP

DIAGRAMING WITH THE UNIFIED MODELING LANGUAGE

UML

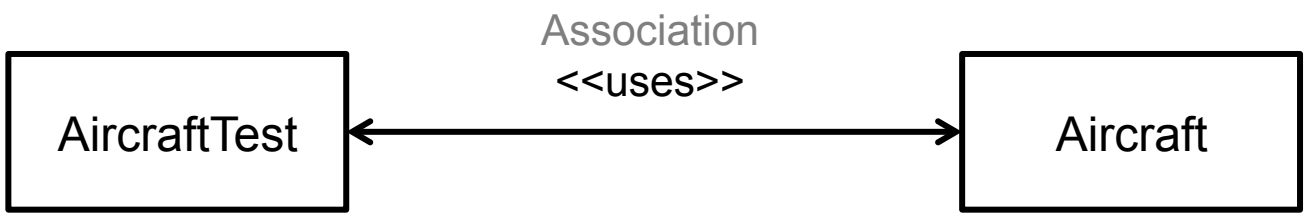
UML CLASS DIAGRAM

A class may be expressed as a UML diagram that shows the primary variables and method calls



UML CLASS DIAGRAM

- **Class diagrams may be drawn to show the relationships between classes (a static view)**
- **Inner class detail is often omitted to stress the class interactions and dependencies.**



THE 'IS-A' AND 'HAS-A' TEST

- **If a class you are designing actually 'is-a' more specific version of another existing class, then you can use inheritance to `extend` from the other class.**
  - Be careful – a changing parent class can break a child
  - Use it judiciously – it tends to violate encapsulation

- **If instead, your class could be said to 'have-a' instance of another class, then you should just own member variables of that other class.**
  - Composition allows you to control method exposure

- **Before extending another class, as yourself this question: Do you want to extend any flaws from that class?**

RELATIONSHIPS

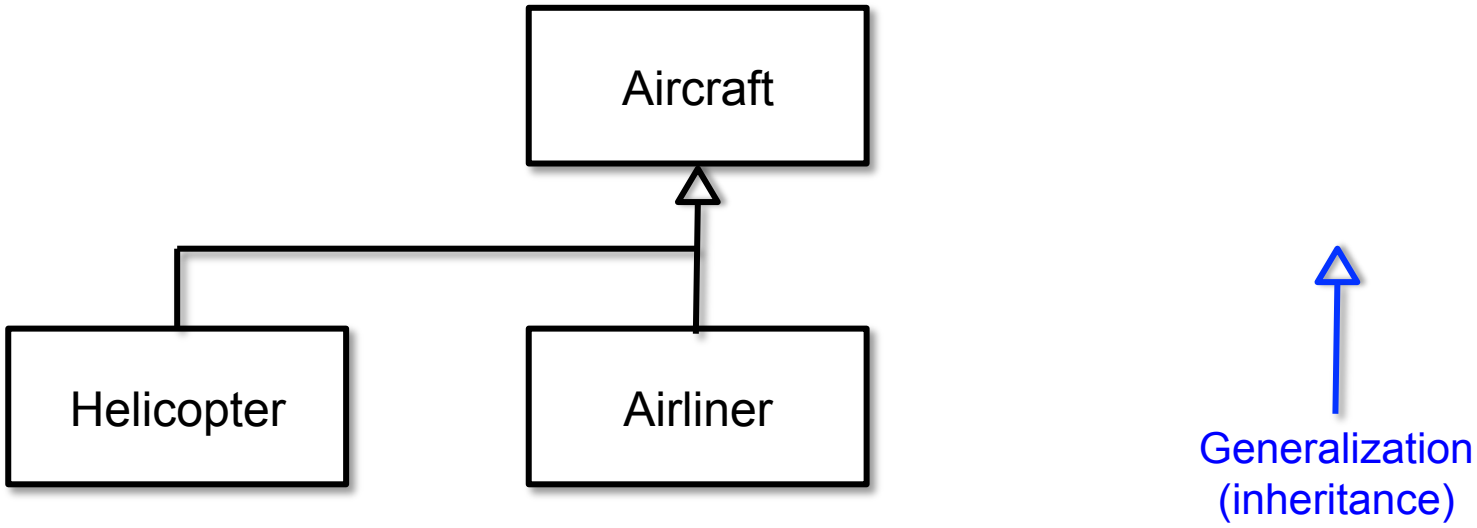
- **Association**

```
direct association(strong ownership)
——>
```

  - Aggregation – A class owns other objects as instance variables
  - This relationship is referred to as "**Has-A**"
- **Generalization**
  - Inheritance – A class inherits variables/methods from a parent class
  - Invoked in Java using the **`extends`** or **`implements`** keywords
  - This relationship is referred to as "**Is-A**"

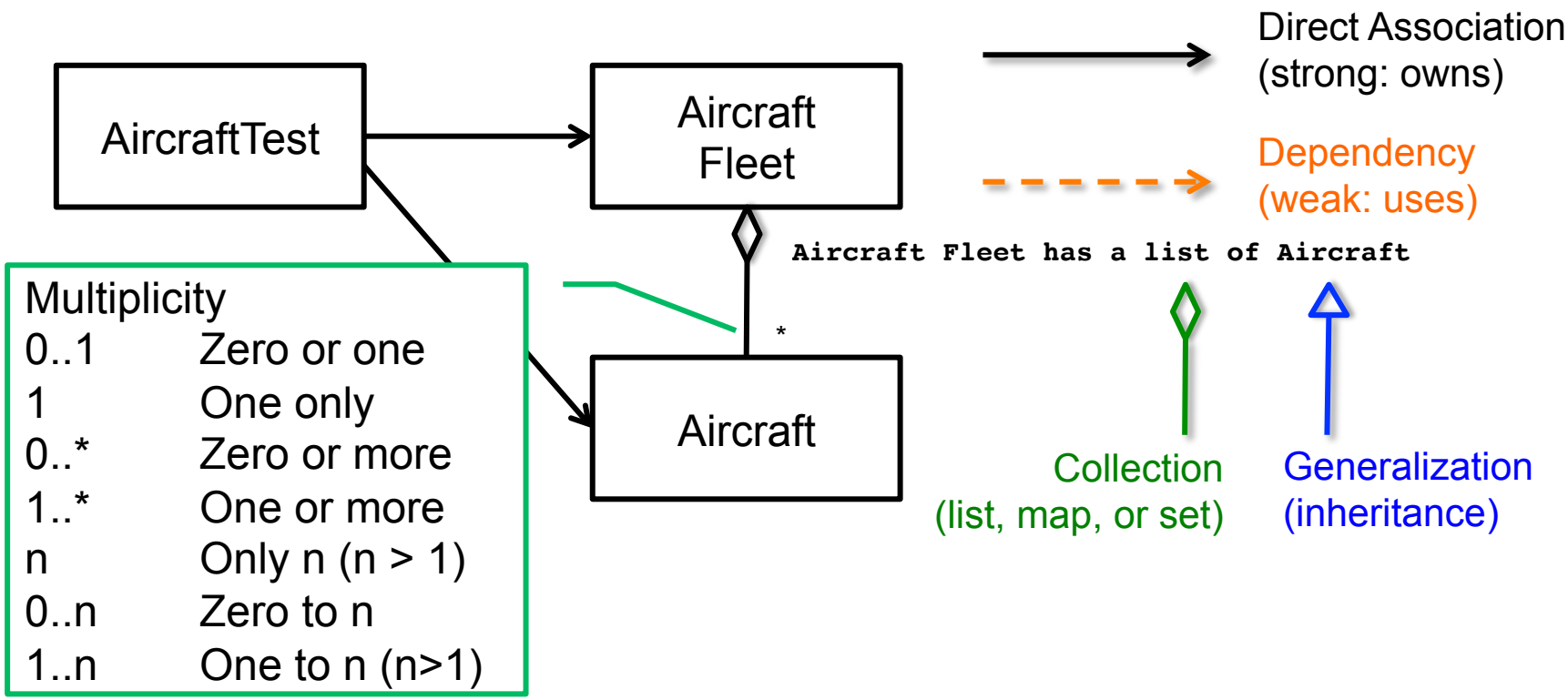
UML CLASS DIAGRAM

- **When inheriting, each inherited class "Is-A" instance of the parent class**



UML CLASS DIAGRAM

- **For an aircraft project , our class structure could be drawn as:**



UML DIAGRAM DEMO

JAVA PACKAGES

PACKAGES

- **To organize and group related software, each class is placed in a `package`.**
  - All classes belong to a package
  - If no package is specified, then the default (global) package is used
  - Java uses the filesystem to manage packages

```
project/src/
            assign2/
                  Aircraft.java
                  AircraftRegistry.java
                  AircraftTest.java
```

PACKAGES

```
project/src/
            assign2/
                    Aircraft.java
                    AircraftRegistry.java
                    AircraftTest.java
```

Package ‘assign2’

- **The `package` statement is placed at the start of each .java file:**

```
package assign2;

class Aircraft {
   public int passengers;
   private double kpl;
…
   public double getKpl() {  // A “getter” method
      return kpl;
   }
…
}
```

PACKAGES (CONT.)

- **After compiling with `javac`, any java source files with a package definition of 'assign2' will have its `.class` file placed in a corresponding subdirectory**

```
project/src/
            assign2/
                    Aircraft.java
                    Aircraft.class
                    AircraftRegistry.java
                    AircraftRegistry.class
                    AircraftTest.java
                    AircraftTest.class
```

Run `javac` from here

- **To run with a package, just use the full *package.Classname* as the target entry point**

```
java assign2.AircraftTest
```

PACKAGES (CONT.)

- **Comingling your source and compiled .class files is inconvenient, so most IDE's will place the .class files in a separate directory of your choosing**

```
project/src/
            assign2/
                    Aircraft.java
                    AircraftRegistry.java
                    AircraftTest.java
project/classes/
                assign2/
                        Aircraft.class
                        AircraftRegistry.class
                        AircraftTest.class
```

javac source defined as 'src'

javac output directory defined as 'classes'

CLASSPATH

- Java uses the CLASSPATH environment variable to locate where .class files reside
- The CLASSPATH variable may be set as part of the java command

```
> java -cp "/proj1/classes;/proj2/classes" assign1.Aircraft
```

First area for classes

Second area for classes

Target *package.class* that has main()

PROTECTED

- **With Java, membership in a package has special benefits**
  - Each class in a package is generally aware of the others, and doesn't need to search to find them (no 'import' needed)
  - In addition to public/private, a third option, **`protected`**, is available

```
class Aircraft {
   public int passengers;
   protected idVal;
   private double kpl;
…
}
```

Open public access

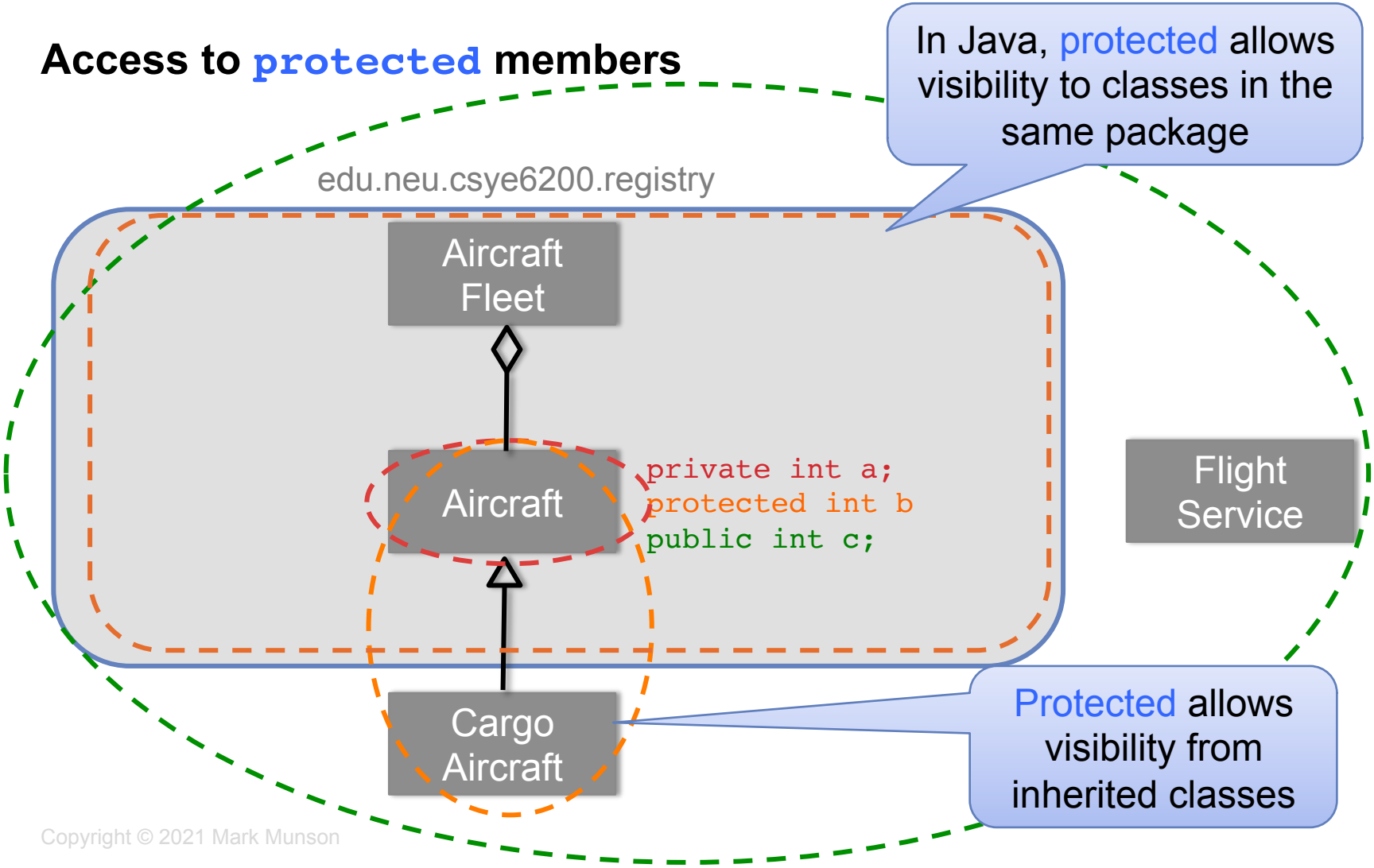
Package family and Inheritance access

Private class-only access

- **In Java, both inherited class and others classes in a package have permission to access `protected` variables and methods.**

FRIENDS AND FAMILY

Access to `protected` members



IMPORT

While other classes in a package can be found automatically, you'll need to help java find classes that exist in other packages

The `import` statement tells java where to look for a class that exists in another package

```
package assign2;

import java.util.ArrayList;

class AircraftRegistry {
   private ArrayList<Aircraft> aList = new ArrayList<Aircraft>();
   private String name;
…
   public AircraftRegistry (String name) {
      this.name = name
}
…
}
```

IMPORT (CONT.)

- The `import` statement has the form

```
import package.classname;
```

- Each class that is used from another package must be defined on an import statement
- If most or all classes from another package need to be imported, you may use an asterisk for the class name

```
import package.*;
```

SCOPE { }

SCOPE { } REVISITED

- Although presented as a collection of statements, scope ({ … }) carries special meaning with regard to variable visibility
- Any time you cross into a new scope, you are effectively creating a new variable space

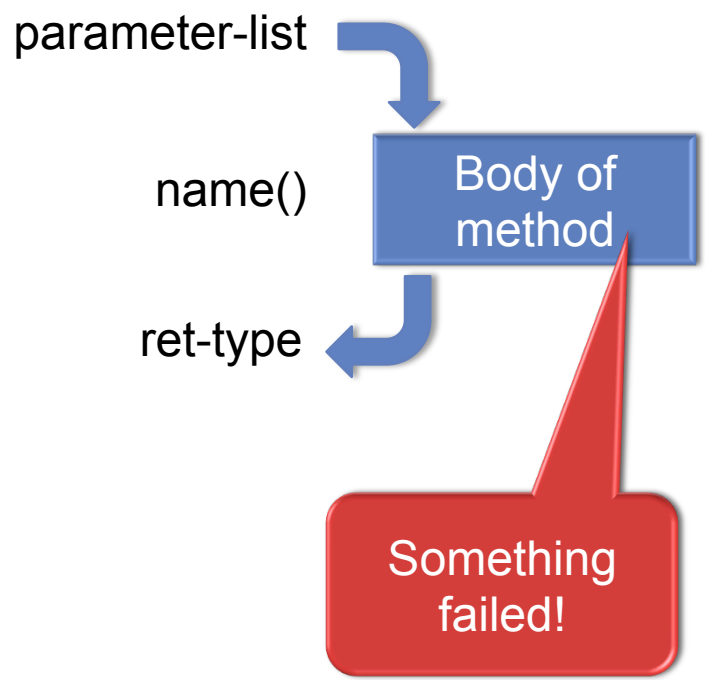
```
class { … }
  void method ( ) { … }
    if ( ) { … }
    { … }
```

- Variables created within a scope, go away when the scope ends

EXCEPTIONS

DEALING WITH ERRORS

- **Error handling has always been a vexing problem**
- **Early attempts to deal with it**
  - Global error variable – requires that you check it often
  - Use ret-type to flag an error – then check a global value to find the type
  - Implement an error method call that checks for a recorded error
- **With Java, there is a better way…**



ERRORS INTO EXCEPTIONS

- In Java each instance of an error is converted into a class called an Exception

```
Exception ex01 = new Exception();
```

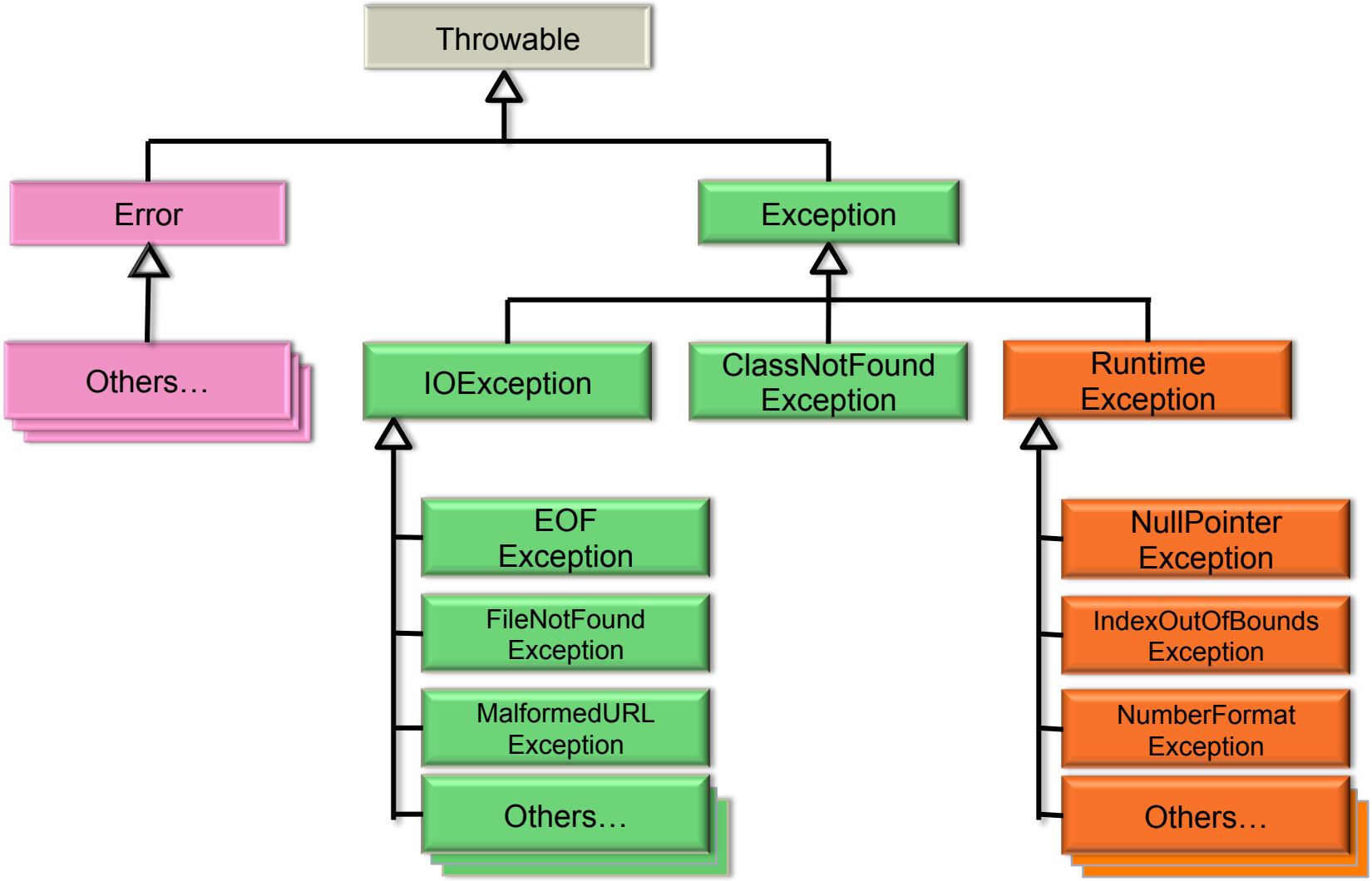
- Exceptions secretly record where you are, and how you arrived there
- Exceptions are organized into a hierarchy of classes, so you can pick one that describes the type of error

```
IOException ex01 = new IOException();
```

- Each Exception allows for a detailed error message

```
IOException ex1 =
  new IOException("Houston, we have a problem");
```

EXCEPTION HIERARCHY



THROWING AN EXCEPTION

Within a method, an Exception is cast using the throw statement

Any methods that aren't handled locally, are flagged using the throws statement

```
private void setName(String name)
  throws IllegalArgumentException {
   if (name.length() == 0)
     throw new IllegalArgumentException("No name");
   …
```

TRY-CATCH

To handle an exception yourself, or to handle one thrown by a routine you called, implement the try-catch block

```
try {
    … statement(s) under test
}
catch (exception-class varName) {
    … actions to take
}
```

TRY-CATCH EXAMPLE

```
public void testExcep() {
  try {
     // Let's make an error (Exception) and throw it
     throw new Exception("This is an error!");
  }
  catch (Exception ex) {
      ex.printStackTrace(); // just print the trace
  }
  finally {
     System.err.println("We caught an error… finally");
  }
}
```

TRY-CATCH-FINALLY

To handle an exception yourself, or to handle one thrown by a routine you called, implement the try-catch block

```
try {
    … statement(s) under test
}
catch (exception-type1 varName) {
    … actions to take if type1 error is caught
}
catch (exception-type2 varName) {
    … actions to take if type2 error is caught
}
finally {
    … final actions to take if any error is received
}
```

EXCEPTION / TRY-CATCH DEMO

FILE I/O INTRO

FILEWRITER

A simple way to write files to disk is to create a `FileWriter` instance

```
import java.io.*;
…
public writeDataToFile(String filename) {
   FileWriter fw;
   try {
      fw = new FileWriter(filename);
      fw.write("first line of text");
      fw.close();
   } catch(IOException ex) {
      System.err.println("IO ERROR received: " + ex.getMessage());
      ex.printStackTrace();
   }
}
```

Imports FileWriter and IOException classes

FILEWRITER (CONT.)

An open FileWriter may be passed to a method()

```
…
public void writeVehicleData(FileWriter fw, Vehicle veh)
    throws IOException {
    fw.write("Line of text");
    fw.write(veh.getMake());
    …
    }
```

FILEWRITER DEMO

FILEREADER

A simple way to read the contents of a file is to create a `FileReader` instance

```
import java.io.*;
…
public void readDataFromFile(String filename) {
    try {
        FileReader fr = new FileReader(filename);
        BufferedReader in = new BufferedReader(fr);
        String str;
        while ((str = in.readLine()) != null) {
            System.out.println("> " + str);
        }
        in.close();
    } catch (FileNotFoundException  | IOException e) {
        e.printStackTrace();
    }
}
```

Imports FileReader and IOException classes

VERSION CONTROL

VERSION CONTROL

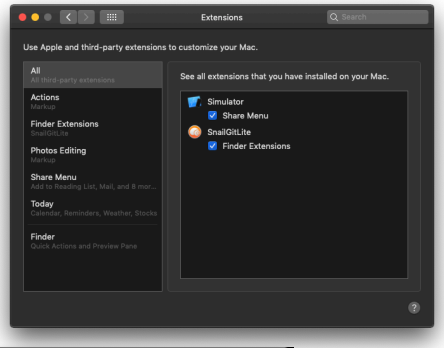
- **RCS – Revision Control System**
  - File based version control
  - Central code repository – check-in, check-out
- **CVS – Concurrent Version Control System**
  - Based on RCS
  - Concurrent check-out
- **SVN – Subversion**
  - Transaction support – commit all or nothing
  - Faster
  - Better handling of binary files
  - Control of file structures
- **Git – Distributed version control**

VERSION CONTROL

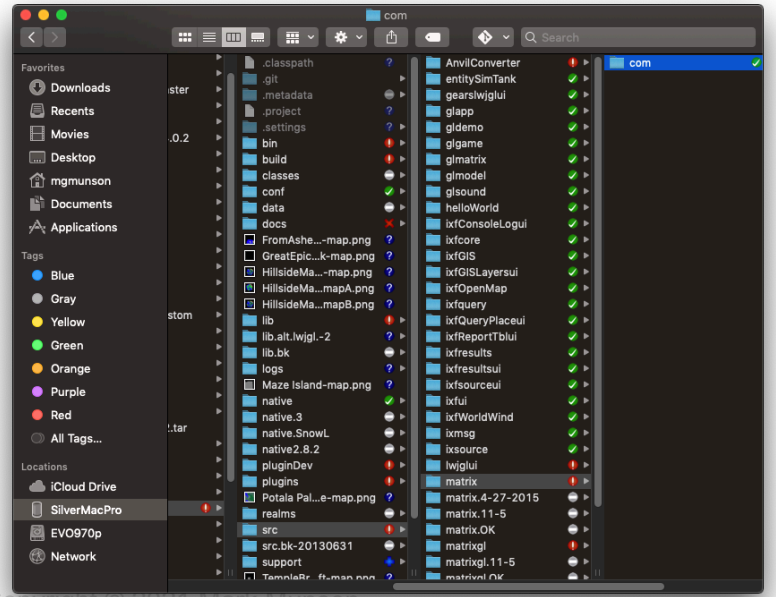
- **Git – Distributed version control**
  - **Offline support:** Each developer has their own repository
    - A local commit may be made, even if internet access isn't available
  - **Atomic:** Commits are handled as full transactions for an entire development tree
  - **Flexible:** Workflow support allows you to use the tool in the way that you want
  - **Clean:** Git only creates a single hidden folder (.git), instead of creating lots of hidden artifacts
  - **Branches:** Code branches are lightweight, instead of cloning the entire codebase

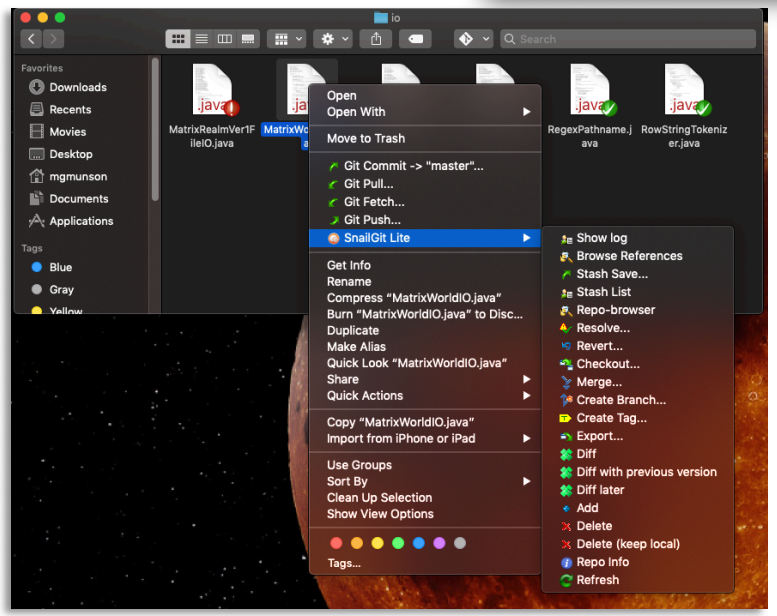
GIT CLIENTS – FILE VIEW

To view GIT file status on your desktop, install a client such as 'TortiseGit' (Windows) or 'SnailGit' (OSX).



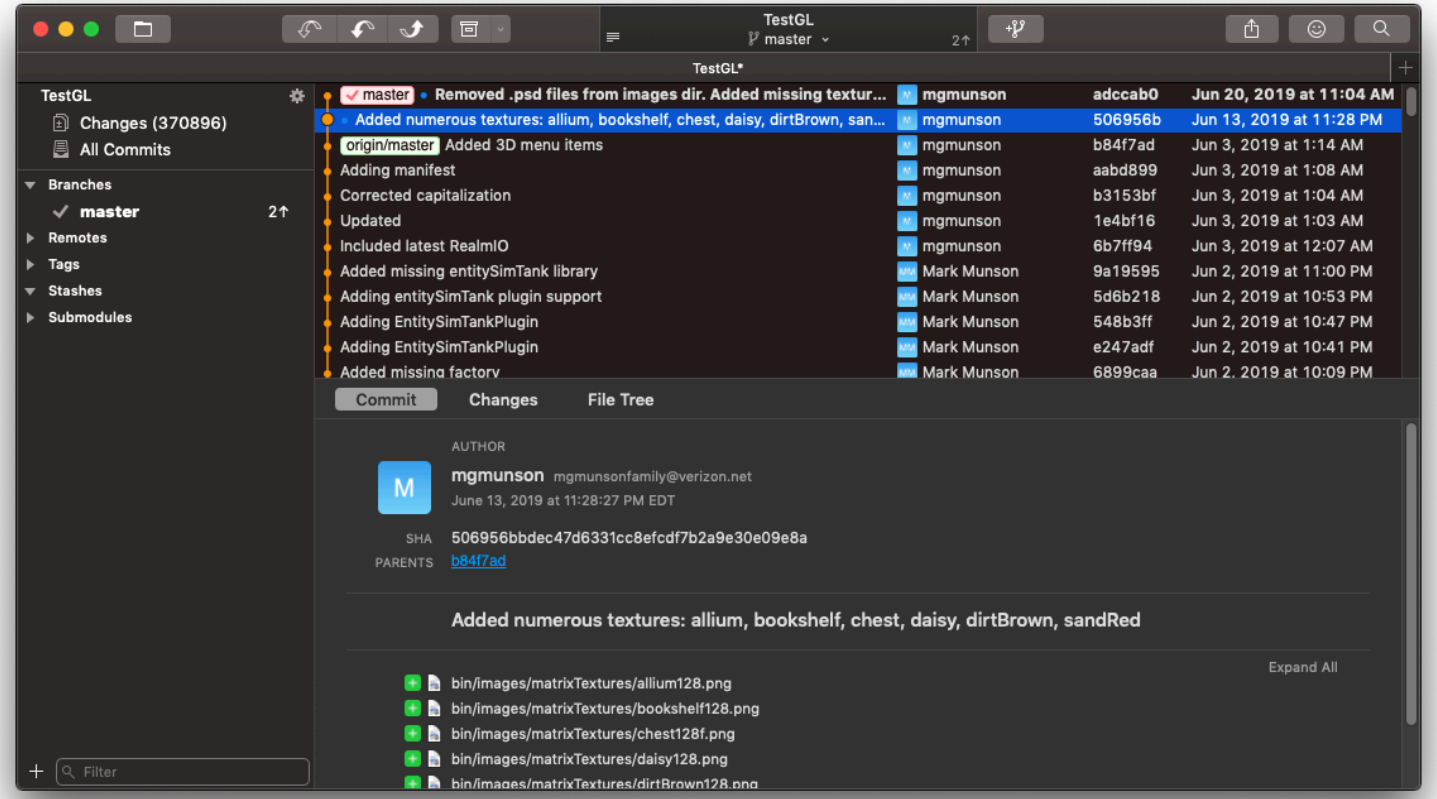
1. **Install the client for the App Store**
2. **Enable in System/Extensions**
3. **Set Repository in the SnailGit Preferences**





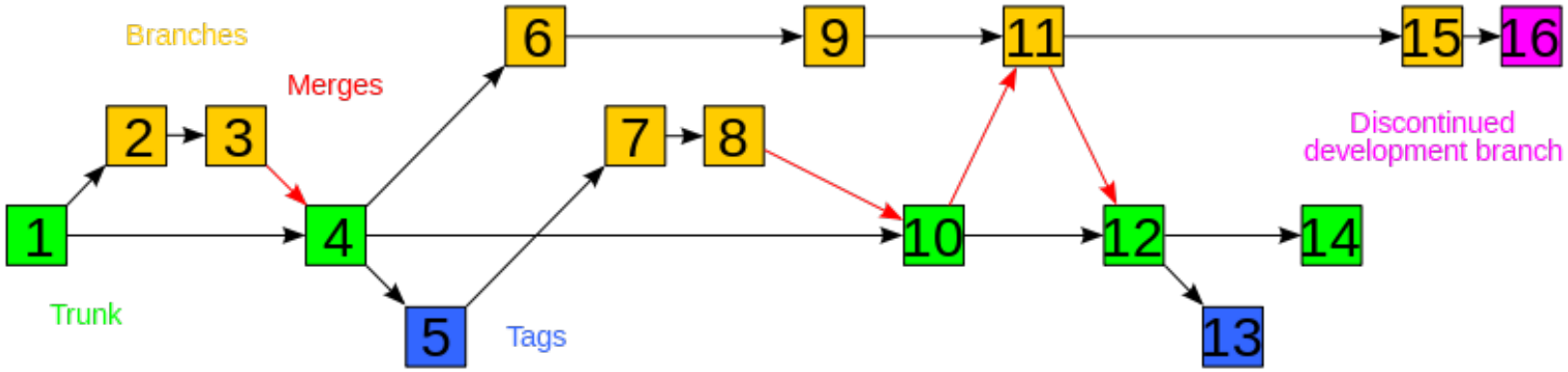
GIT CLIENTS – BRANCH VIEW

For Branch and Merge displays, look to Git client applications such as 'Fork' (git-fork.com).



BRANCH AND MERGE

*



* Wikipedia: Traced by User: Stannered, original by en:User Sami Kerola

GIT
CREATING A LOCAL REPOSITORY

Create a build area (i.e. /proj/CSYE6200)

```
> mkdir CSYE6200
```

Change directories into your build area

```
> cd CSYE6200
```

Create arepository

```
> git init
```

Mark files for addition

```
> touch src
> git add src
```

Commit changes to your local repository

```
> git commit –m "Initial source commit"
```

GIT
REMOTE REPOSITORY

Note: Address supplied by remote repository owner

Add a remote connection

```
> git remote add origin git@localhost:GitRepos/CSYE6200
```

Commit your changes

```
> git commit -a –m "your commit comment"
```

Push your changes to the remote repository

```
> git push origin master
```

Pull changes by others from the remote repository

```
> git pull origin master
```