FINAL EXAM REVIEW

CMPT 270, 2017-18 T1

Course Evaluation

- You will make a difference in how this class is taught next year:

http://evaluation.usask.ca/

CMPT 270 Final Exam

December 13

2:00 PM

PAC GYM - Row 2 to 16 (EVEN)

Reminders

- Bring your Student ID
- No cell phones in the gym

Approx. Structure

- 50 multiple choice (50 marks)
  - ~ 30 concept
  - ~ 20 analyze code
- 2 programming questions (50 marks)
  - 1) implement a class (~10 marks)
  - 2) implement a small system with Animation (in a model-view-controller architecture) (~40 marks)
- 3 hours

Syntax sheets

- Closed book exam
- Three syntax sheets will be provided:
  - Java Basics
  - Java Intermediate
  - Java API Summary

Topics we've covered

- Java Basics
- Objects
- Inheritance
- UML Diagrams
- Data Structures
- Packages
- Exceptions
- Input/Output
- Generics

- Design
  - Procedural Abstraction
  - Object-oriented Design
  - Three-later Architecture
  - Model-View-Control Architecture
- GUIs
- Animation
- Multi-Threading
- Testing

Python vs. Java

Interpreter

- Looks one line at a time
- Memory efficient
- Faster to prototype
- Errors found at runtime

Compiler

- Looks at entire file at a time
- Execution efficient
- Errors found at compile time
- Other errors at runtime

Python vs. Java

Dynamically Typed

- Variables are used without explicit type
- Variables can change type
- Type determined by how the variable is used (operated upon)
- Type errors cause runtime error

Statically Typed

- Variables are declared with explicit type
- Variable type is fixed once declared
- Type checked at compile time
- Type errors cause compiler error
- Limited type conversion

Java Numeric Types

Type	Description	
`int`	The integer type, with range –2,147,483,648 (`Integer.MIN_VALUE`) . . . 2,147,483,647 (`Integer.MAX_VALUE`, about 2.14 billion)	Whole Numbers (no fractions)
`byte`	The type describing a byte consisting of 8 bits, with range –128 . . . 127	
`short`	The short integer type, with range –32,768 . . . 32,767	
`long`	The long integer type, with about 19 decimal digits	
`double`	The double-precision floating-point type, with about 15 decimal digits and a range of about $\pm 10^{308}$	Floating point Numbers
`float`	The single-precision floating-point type, with about 7 decimal digits and a range of about $\pm 10^{38}$	
`char`	The character type, representing code units in the Unicode encoding scheme (see Section 2.6.6)	Characters (no math)

- And the object versions: Integer, Float, Boolean, etc.

Strings

- `Strings` are sequences of characters.
- The length method yields the number of characters in a `String`.
- Use the + operator to concatenate `Strings`; that is, to put them together to yield a longer `String`.
- Use the `next` (one word) or `nextLine` (entire line) methods of the `Scanner` class to read a `String`.
- Whenever one of the arguments of the + operator is a `String`, the other argument is converted to a `String`.
- If a `String` contains the digits of a number, you use the `Integer.parseInt` or `Double.parseDouble` method to obtain the number value.
- `String` index numbers are counted starting with 0.
- Use the `substring` method to extract a part of a `String`

Parsing a value in a string

- **Use Scanner**

```
Scanner inputScanner = new Scanner(stringValue);
int  i = inputScanner.nextInt();
        or
float  f = inputScanner.nextFloat ();
        or
double  d = inputScanner.nextDouble();
        or
String s = inputScanner.next();    // next token (sequence of non-white characters)
```

- **Use a static parse method**

```
int i = Integer.parseInt(stringValue);
                or
float f = Float.parseFloat(stringValue);
                or
double d = Double.parseDouble(stringValue);
```

Note that for the parse methods, the string must contain exactly the value and nothing else.

Methods

- A method is a named sequence of instructions.
- Arguments are supplied when a method is called. The return value is the result that the method computes.
- When declaring a method, you provide a name for the method, a variable for each argument, and a type for the result.
- Method comments explain the purpose of the method, the meaning of the parameters and return value, as well as any special requirements.
- Parameter variables hold the arguments supplied in the method call.

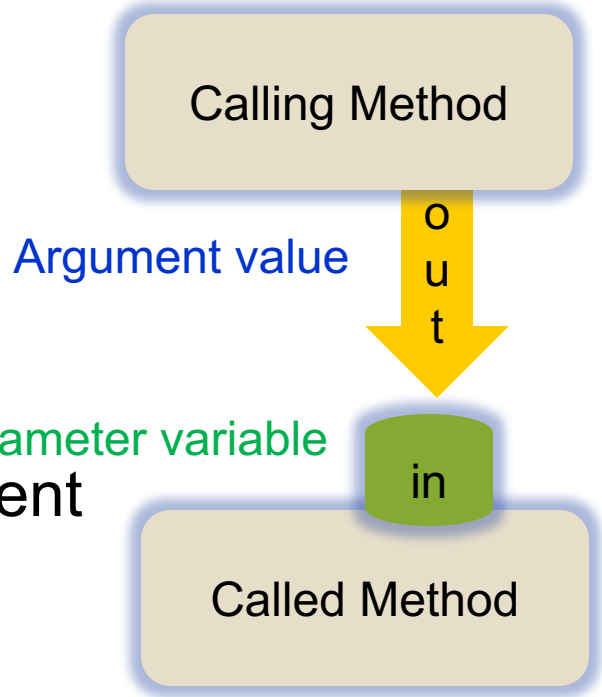
Variable Scope

The scope of a variable is the part of the program in which it is visible.

- Variables can be declared:
  - Inside a method
    - Known as ‘local variables’
    - Only available inside this method
    - Parameter variables are like local variables
  - Inside a block of code { }
    - Sometimes called ‘block scope’
    - If declared inside block { ends at end of block }
  - Outside of a method
    - Sometimes called ‘global scope’
    - Can be used (and changed) by code in any method
- How do you choose?

Parameter Passing

- Parameter variables receive the argument values supplied in the method call
  - They both must be the same type
- The argument value may be:
  - The contents of a variable
  - A ‘literal’ value (2)
  - aka. ‘actual parameter’ or argument
- The parameter variable is:
  - Declared in the called method
  - Initialized with the value of the argument value
  - Used as a variable inside the called method
  - aka. ‘formal parameter’



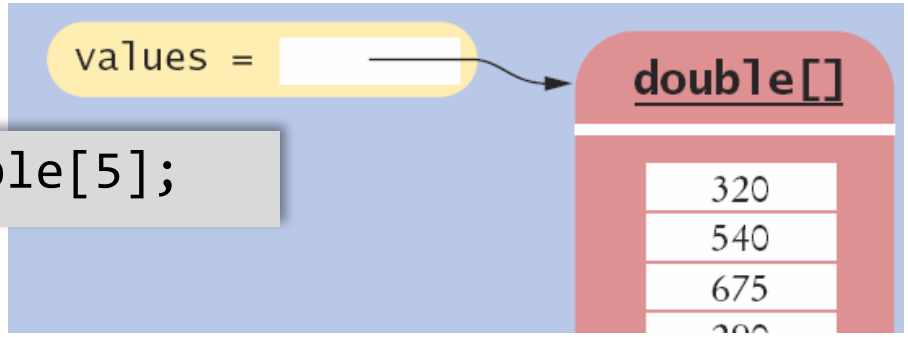
Variables and Methods

- Instance Variable
- Instance Method
- Private Instance Variable
- Static Variable
- Accessor Method
- Mutator Method

Object References

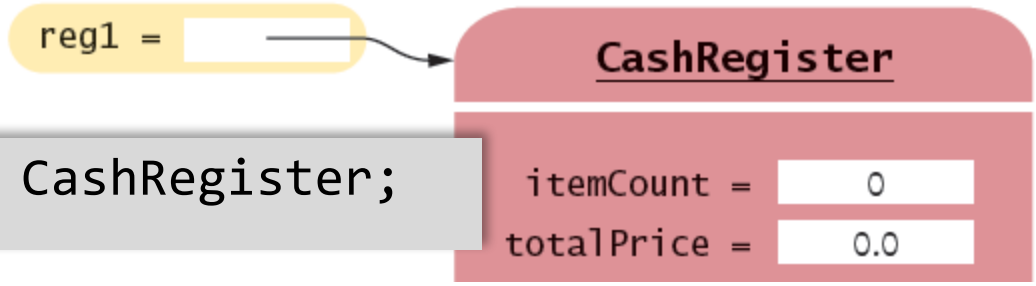
- Objects are similar to arrays because they always have reference variables
  - Array Reference

```
double[] values = new double[5];
```



  - Object Reference

```
CashRegister reg1 = new CashRegister;
```

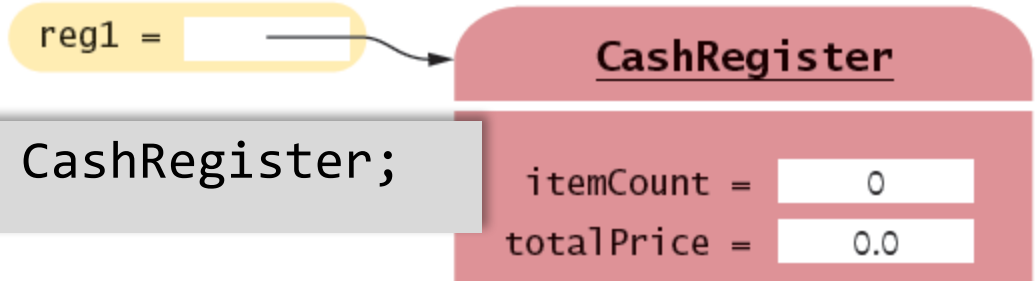


An object reference specifies the *memory location* of the object

Shared References

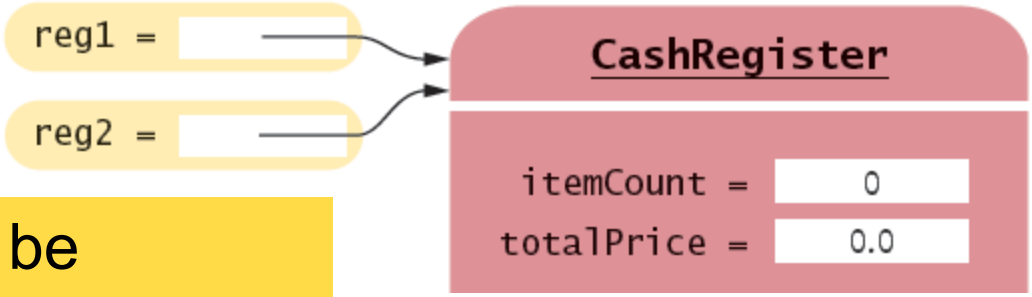
- Multiple object variables may contain references to the same object.
  - Single Reference

```
CashRegister reg1 = new CashRegister;
```



  - Shared References

```
CashRegister reg2 = reg1;
```

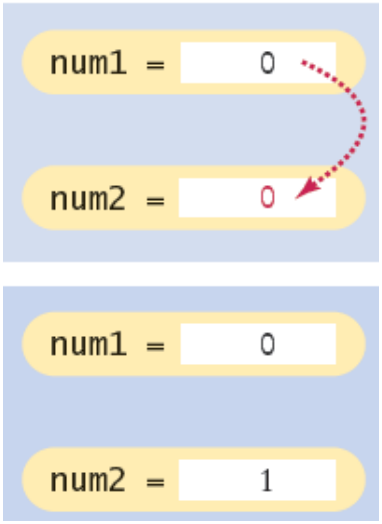


The internal values can be changed through either reference

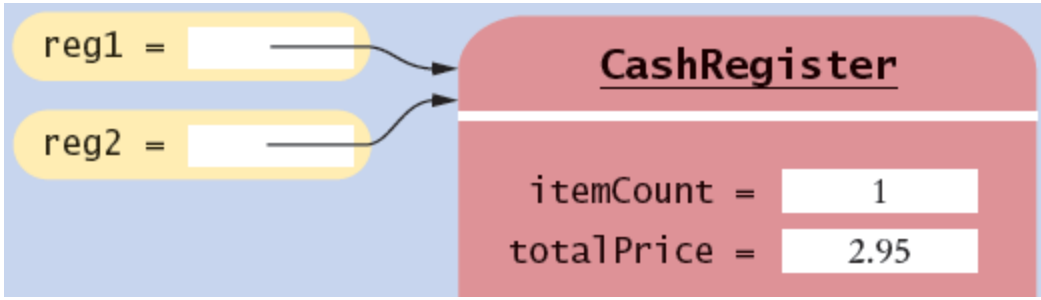
Primitive versus Reference Copy

- Primitive variables can be copied, but work differently than object references
  - Primitive Copy
    - Two locations
  - Reference Copy
    - One location for both

```
int num1 = 0;
int num2 = num1;
num2++;
```




```
CashRegister reg1 = new CashRegister;
CashRegister reg2 = reg1;
reg2.addItem(2.95);
```



Why? Primitives take much less storage space than objects!

Inheritance

- A subclass inherits data and behavior from a superclass.
- You can always use a subclass object in place of a superclass object.
- A subclass inherits all methods that it does not override.
- A subclass can override a superclass method by providing a new implementation.

Overriding Methods

- An overriding method can extend or replace the functionality of the superclass method.
- Use the reserved word `super` to call a superclass method.
- Unless specified otherwise, the subclass constructor calls the superclass constructor with no arguments.
- To call a superclass constructor, use the `super` reserved word in the first statement of the subclass constructor.
- The constructor of a subclass can pass arguments to a superclass constructor, using the reserved word `super`.

Polymorphism

- A subclass reference can be used when a superclass reference is expected.
- Polymorphism ("having multiple shapes") allows us to manipulate objects that share a set of tasks, even though the tasks are executed in different ways.
- An `abstract` method is a method whose implementation is not specified.
- An `abstract` class is a class that cannot be instantiated.

Polymorphism

- Method calls *are always*
  - *determined by the type of the actual object,*
  - **not** the type of the variable containing the object reference

- multiple objects to carry out a task;
- each object does so in its own way

Access

- `private` Access
  - Only access by the methods of the object's class
- `protected` Access
  - The superclass can declare an instance variable as `protected` instead of `private`
  - `protected` data in an object can be accessed by the methods of the object's class and all its subclasses.
  - But it can also be accessed by all other classes in the same package!
- `public` Access
- `no modifier`

Interfaces

- The Java `interface` type contains the return types, names, and parameter variables of
- Unlike a class, an `interface` type provides no implementation.
- By using an interface type for a parameter variable, a method can accept objects from many classes.
- The `implements` reserved word indicates which interfaces a class implements.
- Implement the `Comparable` interface so that objects of your class can be compared, for example, in a sort method.

Testing Terminology

- Error
- Fault
- Failure

Test case

- A test case consists of
  - a set on input values
  - a set of preconditions that must hold
  - a set of expected outputs

Approaches to Testing

- Human testing: people reading the documents/code
- Black-box testing: tests based on the specification only, ignores actual code
- White-box testing: tests based on the code used to implement the system
- Object-oriented testing: tests particularly suited to systems implemented by an object-oriented language

Black-box Testing

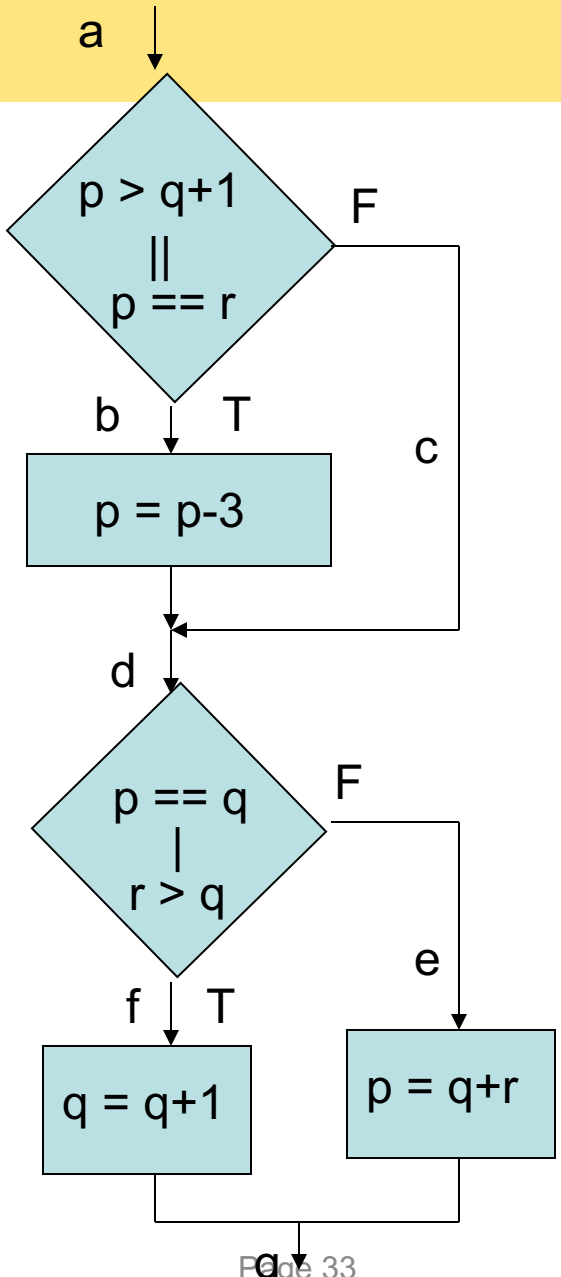
- Based on the specification of what the program should do (not how)
  - Need a precise specification of what the program should do
  - Specification is based on preconditions, postconditions, and contracts
- The tests should be written before the code.
- Techniques:
  - Outcome testing
  - Boundary-value testing
  - Equivalence-class testing

White-box Testing

Testing based on looking at the code

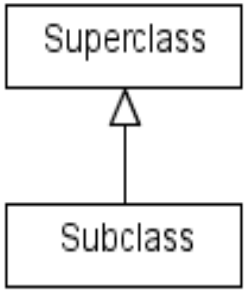
1. Statement coverage
   - Execute each statement at least once.
   - e.g., paths: abdfg and abdeg
   - data: p=5, q=2, r=4
     p=5, q=0, r=-1
2. Branch coverage
   - Execute every branch at least once.
   - e.g., paths: abdfg and acdeg
   - data: p=5, q=2, r=4
     p=3, q=2, r=1
3. Condition coverage
   - Try the true and false cases for each condition.
   - For the pair of if conditions, try all combinations of true and false.
   - TT, TF. FT, FF
     Need at least 4 paths.
     Try each value for each boolean expression (8 cases).
4. Path coverage
   - Execute every path.
   - Paths: abdfg, abdeg, acdfg, acdeg

Note: path coverage does not imply condition coverage, and condition coverage might not imply path coverage

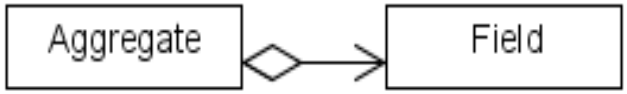


More UML relationships:

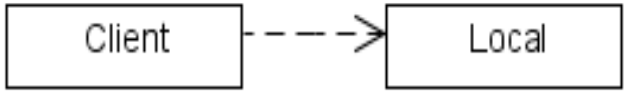
inheritance
Subclass extends Superclass
Superclass is the parent
Subclass is the child



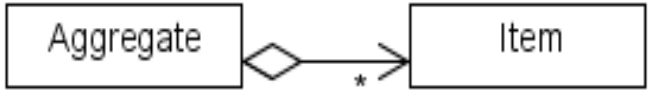
aggregation
Aggregate has a field called Field



dependency (uses)
Client uses Local
e.g., Local is the type of a local variable
e.g., Local is the type of a parameter



container aggregation
Aggregate has a field that is
is a container of Item



Standard UML does not have an arrowhead for aggregation, only the diamond.

Abstraction

- eliminate irrelevant details
- ignore relevant details that aren't needed for the current task
- generalize the problem to handle more situations and allow its reuse

Procedural abstraction

- To use a procedure, you need to know:
  - how to invoke the procedure
  - when the procedure can be used (Pre-condition)
  - what does it do or return (Post condition)

Cohesion (1)

- A class should represent a single concept
- The public interface of a class is **cohesive** if all of its features are related to the concept that the class represents

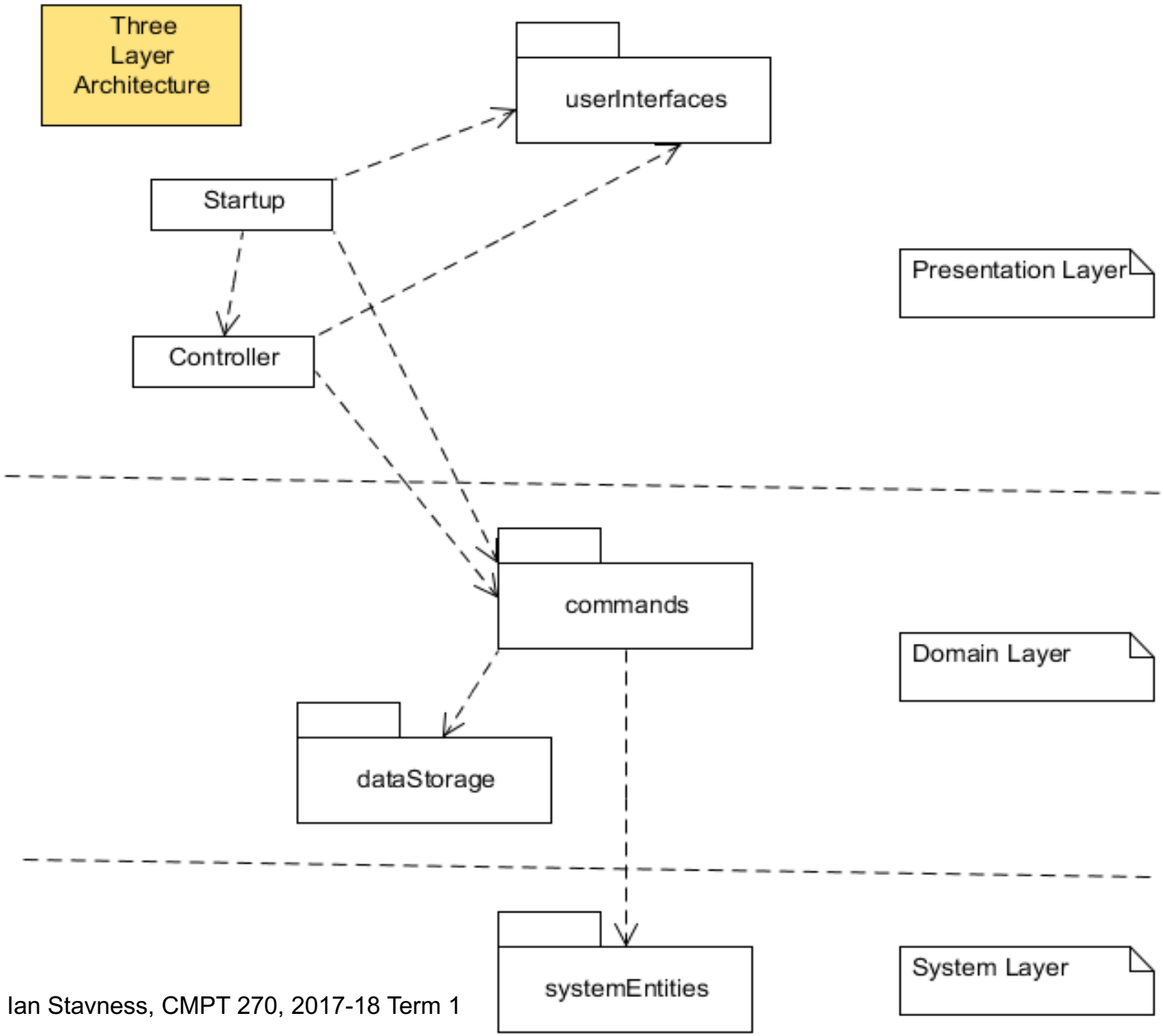
Coupling (1)

- If many classes depend on each other, the **coupling** between classes is high
- Good practice: minimize coupling between classes
  - Change in one class may require update of all coupled classes
  - Using a class in another program requires using all classes on which it depends

Object-Oriented Design

Class Relationships and UML Diagrams

- A class depends on another class if it uses objects of that class.
- It is a good practice to minimize the coupling (i.e., dependency) between classes.
- A class aggregates another if its objects contain objects of the other class.
- Inheritance (the *is-a* relationship) is sometimes inappropriately used when the *has-a* relationship would be more appropriate.
- Aggregation (the *has-a* relationship) denotes that objects of one class contain references to objects of another class.


Three Layer Architecture
userInterfaces
Startup
Presentation Layer
Controller
commands
Domain Layer
dataStorage
systemEntities
System Layer

Command

- Check the parameters for valid values
- Carry out the command
  - If successful

    store the results in appropriate fields
  - Else

    store the error message in its field

Controller main loop

- Using the interface obtain the id of the next command
- Determine the command selected
- Using the interface obtain any needed parameter values of the correct type
- Create the command
- Execute the command
  - If successful

    access the results from the command object

    handle the results, e.g., display the results using the interface
  - Else

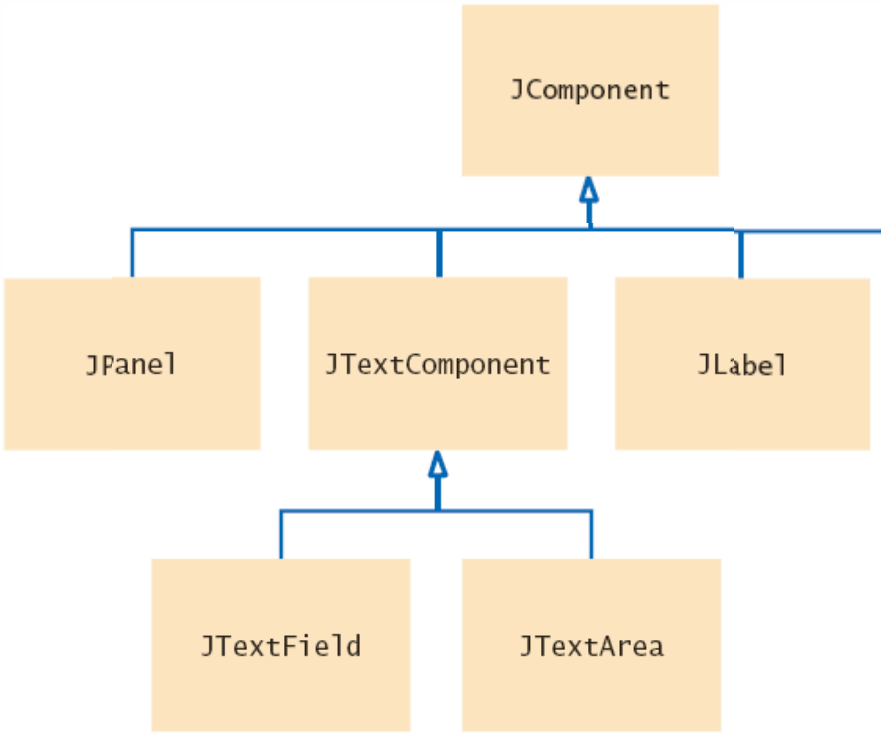
    access the error message, and display it using the interface

Frames and Components

- To show a frame, construct a `JFrame` object, set its size, and make it visible.
- Use a `JPanel` to group multiple user-interface components together.
- Declare a `JFrame` subclass for a complex frame.

JComponents

- JComponent
- JPanel
- JTextComponent
- Jlabel
- JButton



Layout Management

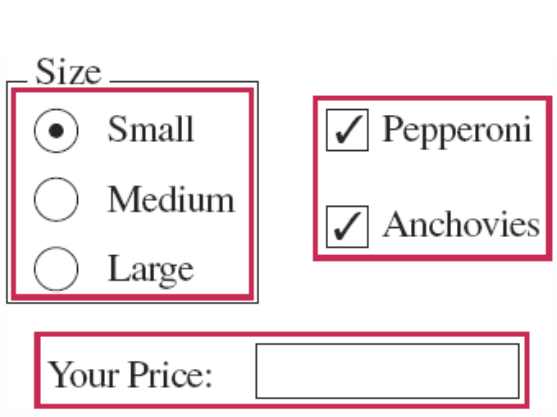
- Each container has a layout manager that directs the arrangement of its components
- Three useful layout managers are:

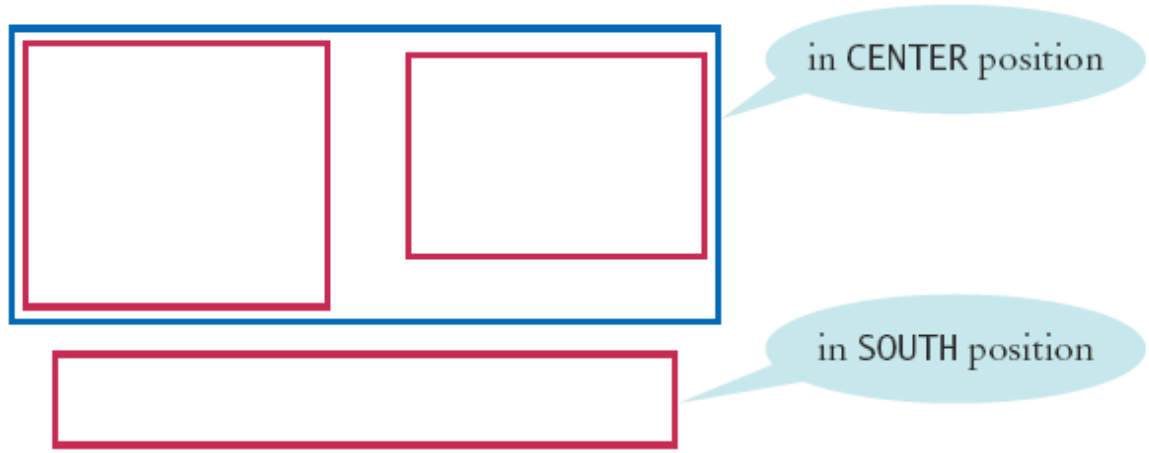
  1) Border layout
  2) Flow layout
  3) Grid layout

Steps to Design a User Interface

3) Identify layouts for each group.
   - For horizontal components, use flow Layout
   - For vertical components, use a grid layout with one column

4) Group the groups together.
   - Look at each group as one blob, and group the blobs together into larger groups, just as you grouped the components in the preceding step





Events and Handlers

- User-interface events include key presses, mouse moves, button clicks, menu selections, and so on.
- An event listener belongs to a class created by the application programmer.
  - Its methods describe the actions to be taken when an event occurs.
  - Event sources report on events. When an event occurs, the event source notifies all event listeners.
- Attach an `ActionListener` to each button so that your program can react to button clicks.
- Methods of an inner class can access variables from the surrounding class.

Dialog Box

- Used to display a message or obtain input from the user
- A new window pops up overtop of the project window
- Suspends the project until an option in the box is selected

- **Message display**

```
import javax.swing.JOptionPane;
JOptionPane.showMessageDialog(null, message);  // invoke a static method
                                               // in class JOptionPane
```

- **Input a String value**

```
import javax.swing.JOptionPane;
String  stringValue  = JOptionPane.showInputDialog(null, promptMessage);
```

- Note that if a JOptionPane is used, the system creates a new user interface thread. To halt that thread, you need to add the following line as the last statement executed of your program: System.exit(0);

More elaborate message

```
JOptionPane.showMessageDialog(
                                    null,                    // parent component
                                    “My message",            // the message to display
                                    “My window title",       // the title for the window
                                    JOptionPane.PLAIN_MESSAGE);   // type of message
```

The message types are one of the static **int** constants of JOptionPane:

JOptionPane.*ERROR_MESSAGE*
JOptionPane.*INFORMATION_MESSAGE*
JOptionPane.*WARNING_MESSAGE*
JOptionPane.*QUESTION_MESSAGE*
JOptionPane.*PLAIN_MESSAGE*

Animation with Timer vs. Thread

- `javax.swing.Timer`
  - Can generate a series of events at even time intervals
  - Specify the frequency of the events and an object of a class that implements the `ActionListener` interface
- Animation with Threads

Running Threads

- A thread is a program unit that is executed concurrently with other parts of the program.
- The `start` method of the `Thread` class starts a new thread that executes the `run` method of the associated `Runnable` object.
- The `sleep` method puts the current thread to sleep for a given number of milliseconds.
- When a thread is interrupted, the most common response is to terminate the `run` method.
- The thread scheduler runs each thread for a short amount of time, called a time slice.

Terminating Threads

- A thread terminates when its run method terminates.
- The `run` method can check whether its thread has been interrupted by calling the `interrupted` method.

Race Conditions

- A race condition occurs if the effect of multiple threads on shared data depends on the order in which the threads are scheduled.
- Synchronizing Object Access:
  - By calling the `lock` method, a thread acquires a `Lock` object. Then no other thread can acquire the lock until the first thread releases the lock.
  - Or use **synchronized** keyword

Bouncing Ball: Threads & Sync

- Now, how many threads are used in the ThreadAnimation program?
  - main thread
  - event dispatch thread
  - new thread

Is there conflict between the three threads?

main

event dispatch thread

handles mouse clicks

updates the ball position

issues repaint( ) instructions

new thread

updates the ball position

issues repaint( ) instructions

Problem:
•Two threads updating and accessing the same data.
•This is called a **Race Condition**: the results depend upon the scheduling of the threads.

Space Invaders: Threads & Sync

- Main thread
  - terminates when the welcome window is made visible
- Game thread
  - runs the game
  - executes gameChanged() method in the observers
- Event-dispatch thread
  - handles events
    - GUI tasks of the non-game windows
    - player actions (key presses) in the game window
  - handles paintComponent (an event is used to invoke paintComponent)
    - Solves race condition that would occur if a component were updated
    - in the event-dispatch thread, while it was being painted in another thread.
    - **Note**: this is why you don't invoke paintComponent() directly!.

Generic Classes and Type Parameters

- In Java, generic programming can be achieved with inheritance or with type parameters.
- A generic class has one or more type parameters.
- Type parameters can be instantiated with class or interface types.
- Type parameters make generic code safer and easier to read.

Generic Classes and Interfaces

- Type variables of a generic class follow the class name and are enclosed in angle brackets.

```
public class Pair<T, S>
```

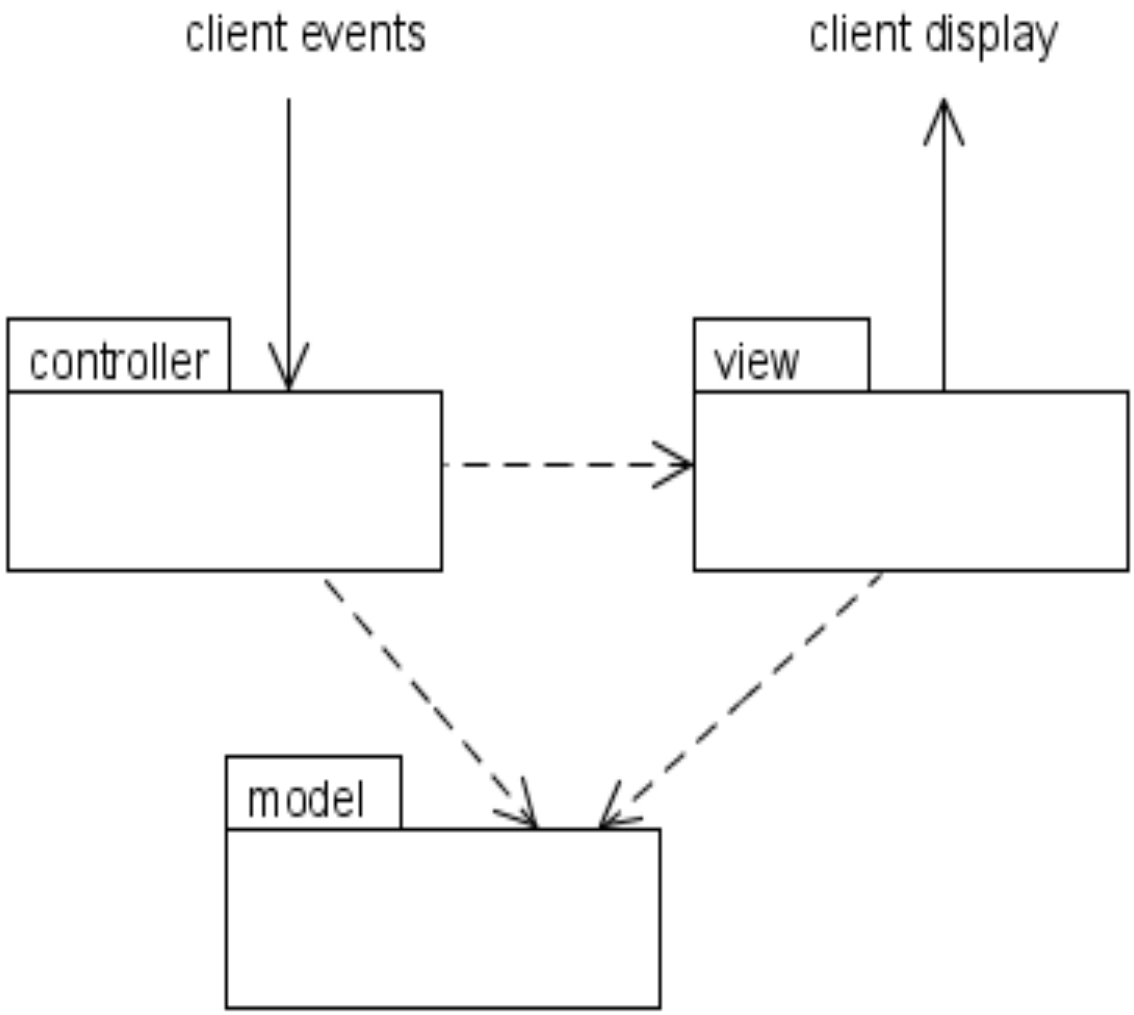
- Use type parameters for the types of generic instance variables, method parameters, and return values.

Generic Methods

- A generic method is a method with a type parameter.
- Supply the type parameters of a generic method between the modifiers and the method return type:

```
public static <E> void print(E[] a)
```

Basic Model-View-Controller



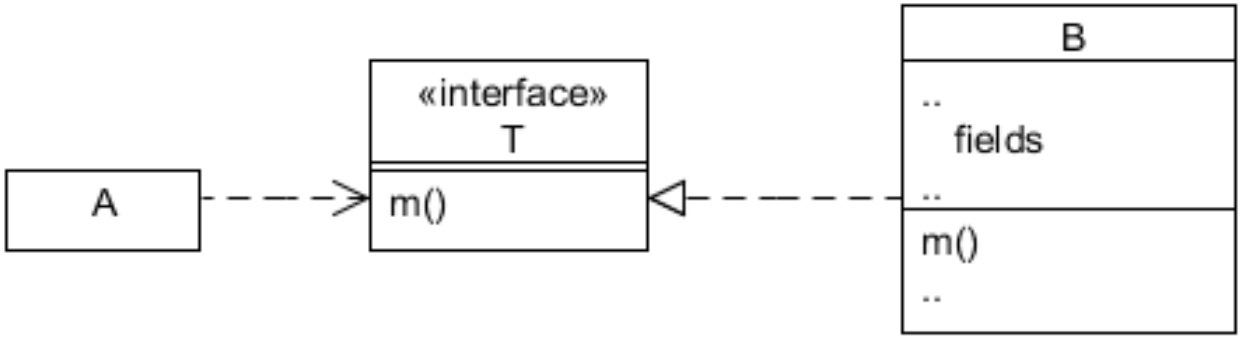
GUI version: where is the controller?

- In a GUI application:
  - many of the controller tasks are done by listeners
  - often have a different listener for each command
  - or the switch statement is in the listener
- Therefore, the Controller code is intermixed with the Interface code (within the Presentation layer).
- But we can separate the Controller from the Interface by an OO-design scheme called the Model-View-Controller architecture

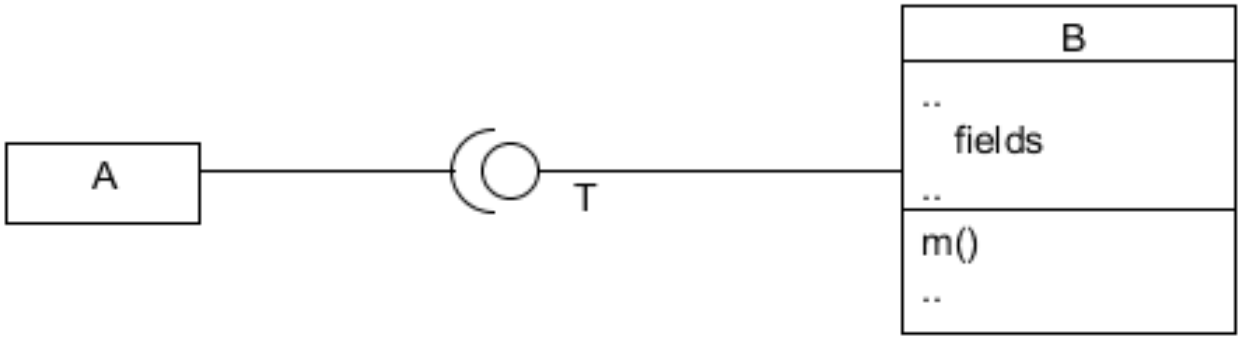
Minimize coupling: use interfaces

- If A only needs method **m()** of B, how can this strong coupling be reduced?

  Define an interface T with the features of B needed by A.

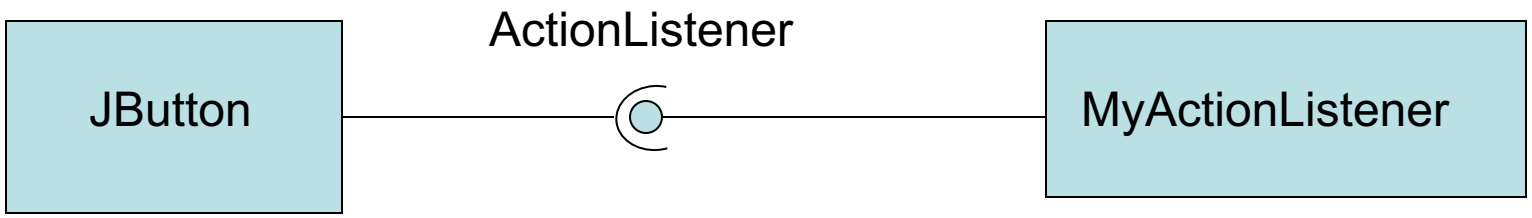


Alternate notation:

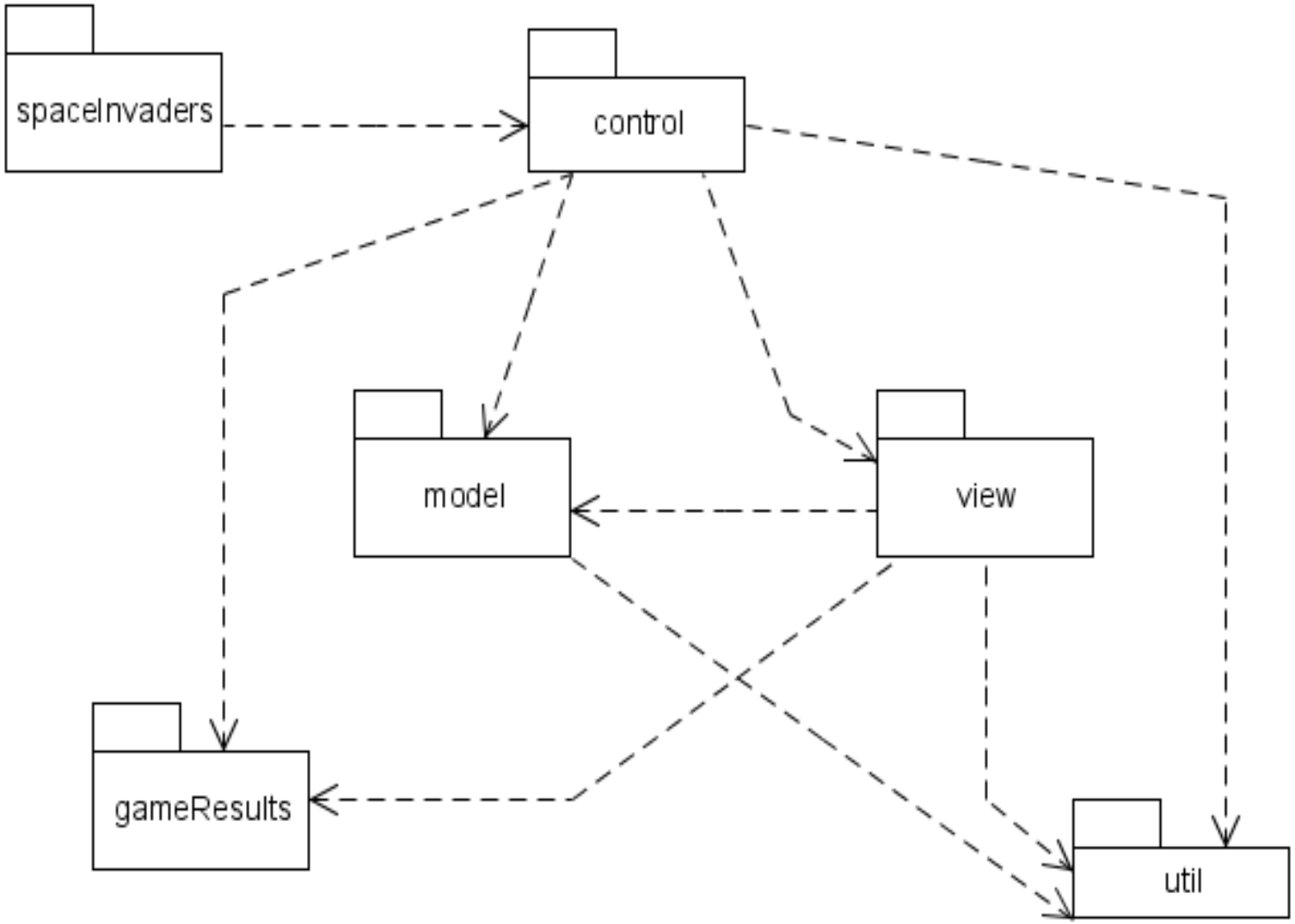


Example: JButton

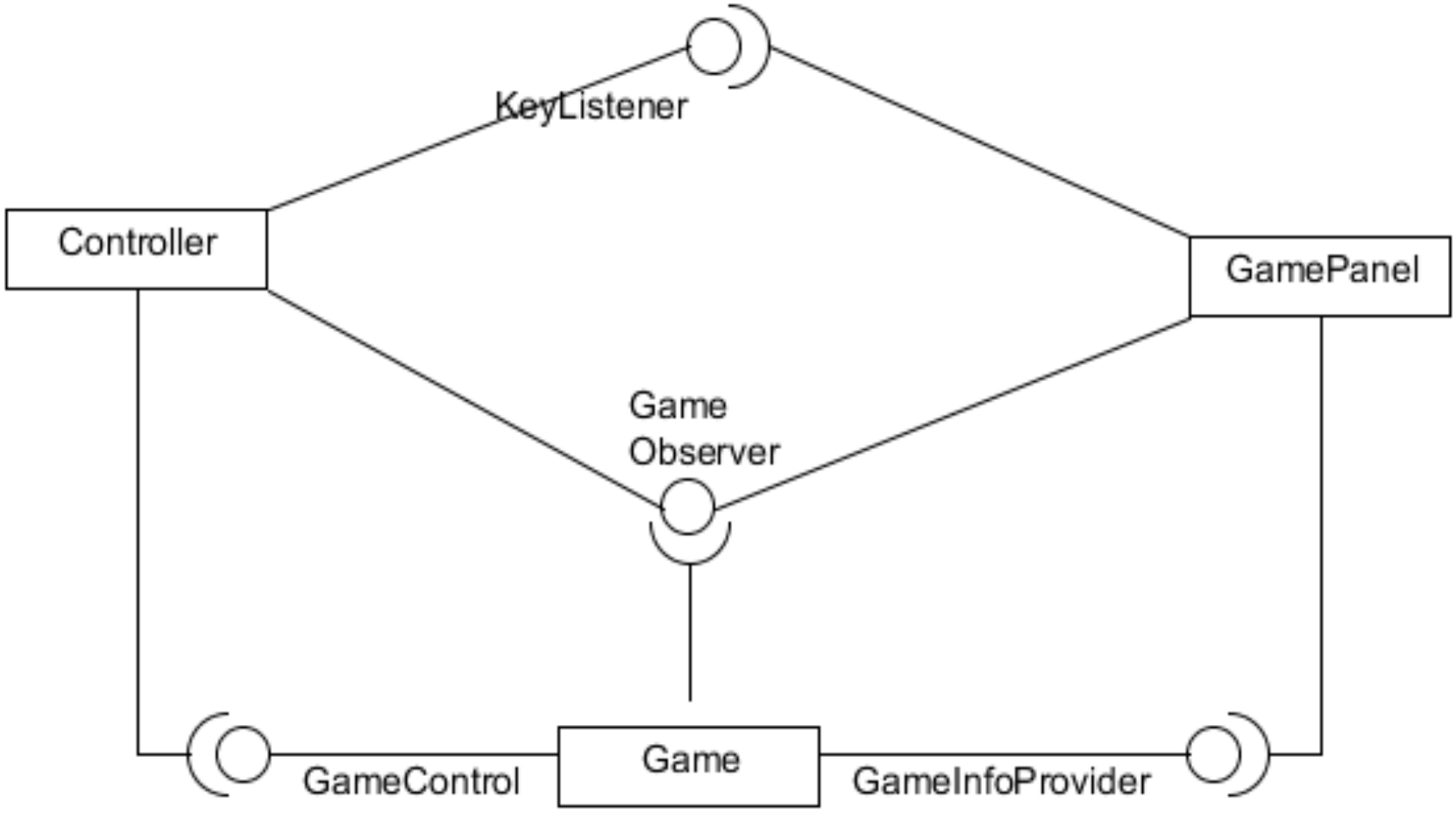
- The relationship is that some class, JButton in this example, interacts with an object of an interface type, while some other class provides the implementation of the interface.
- This interaction is common enough that there is special notation for it in a UML diagram.



SpaceInvaders: Package interaction



SpaceInvaders: High-level class diagram



Good luck!!

- Hope to see you in Spinks
- And in Computer Graphics (485)

- Always looking for students interested in joining my lab for summer and grad school