CSSE 220

A Software Engineering Technique:
(Class Diagrams)

Download FirstOODesignPractice from SVN

Designing Classes

- Programs typically begin as abstract ideas
- These ideas form a set of abstract requirements
- We must take these abstract requirements, use piecewise elaboration and refinement until specifications emerge
    - Then models
    - … concrete implementation

Software Process

- Many different options
  - Waterfall, Spiral, Iterative, etc.
- For this class, we'll follow a much simpler process than these:
  - Design
  - Development/Test
- These are not mutually exclusive, but a good order to start with, then **elaborate** and **refine**
  - Feel free to write tests before development if you like TDD ☺

Design Tools

- Class Diagramming

Tools of the Trade

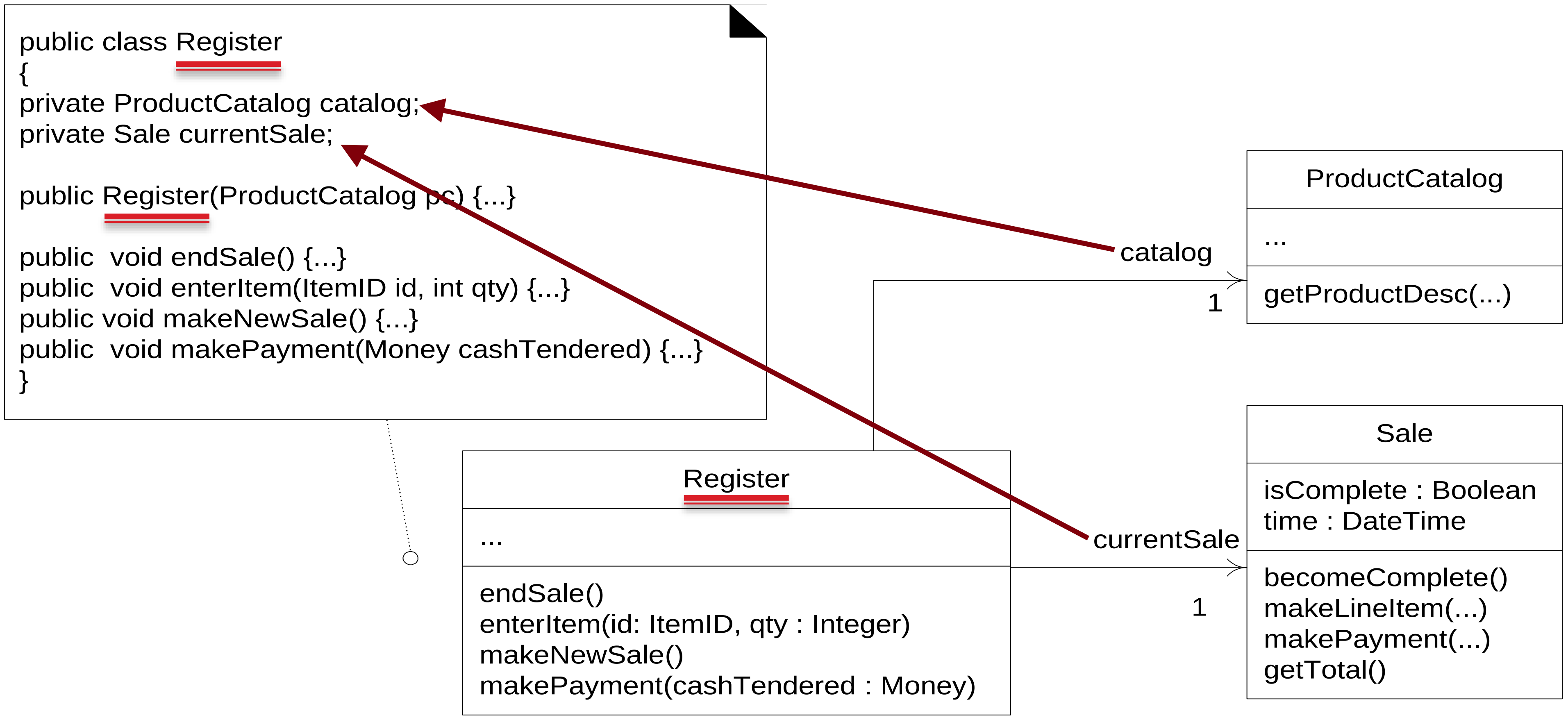
- Class Diagrams (UML)
- UML – Unified Modeling Language
  - Language unspecific
  - provides guidance as to the order of a team’s activities
  - specifies what artifacts should be developed
  - directs the tasks of individual developers and the team as a whole
  - offers criteria for monitoring and measuring a project’s products and activities

According to UML-Diagrams.org

- The **Unified Modeling Language™** (**UML®**) is a standard visual modeling language intended to be used for
  - modeling business and similar processes,
  - analysis, design, and implementation of software-based systems

UML is a common language for business analysts, software architects and developers used to describe, specify, design, and document existing or new business processes, structure and behavior of artifacts of software systems.

Diagramming Classes

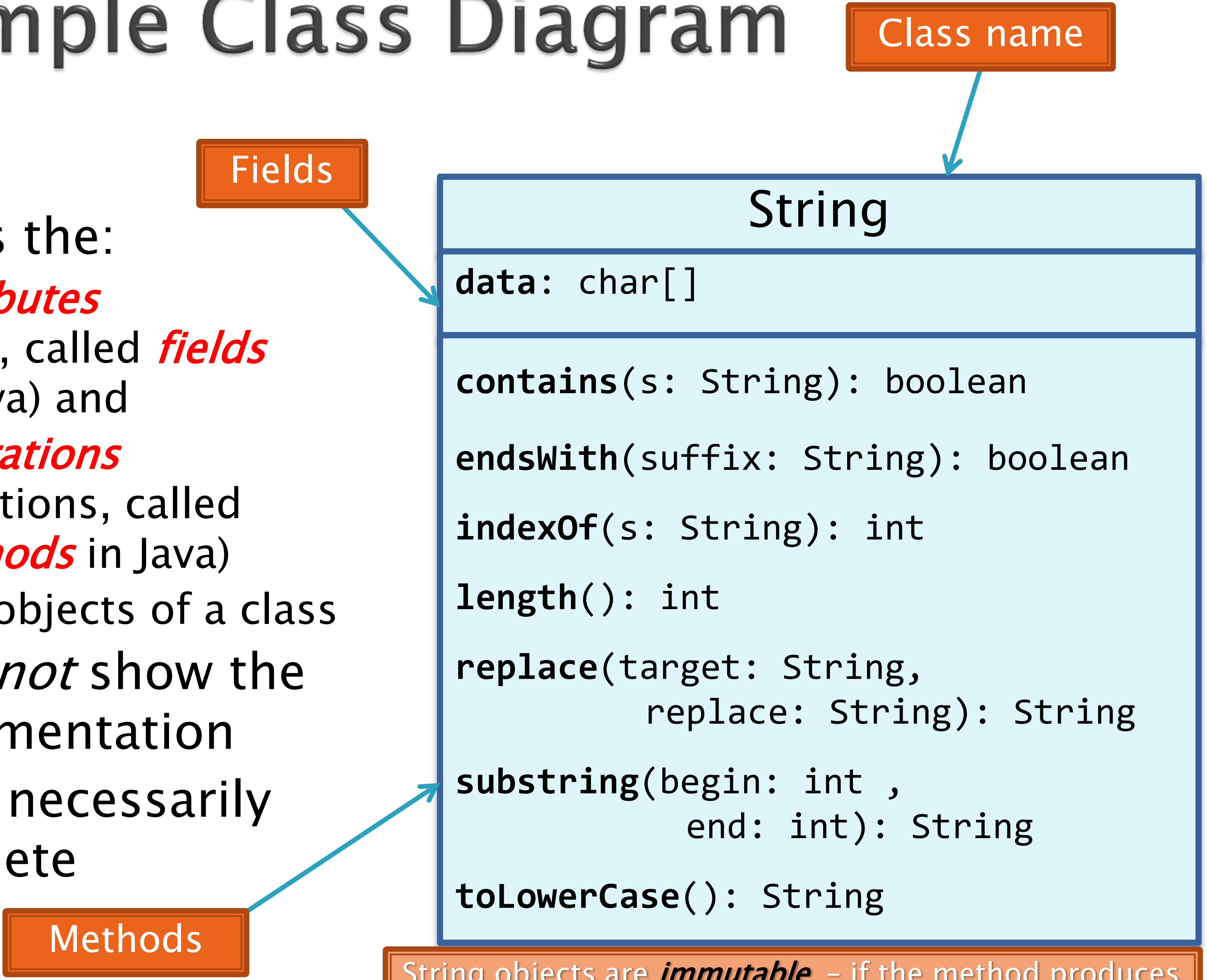


```
public class Register
{
private ProductCatalog catalog;
private Sale currentSale;

public Register(ProductCatalog pc) {...}

public  void endSale() {...}
public  void enterItem(ItemID id, int qty) {...}
public void makeNewSale() {...}
public  void makePayment(Money cashTendered) {...}
}
```

Example Class Diagram

- Shows the:
  - ***Attributes*** (data, called ***fields*** in Java) and
  - ***Operations*** (functions, called ***methods*** in Java)

  of the objects of a class
- Does *not* show the implementation
- Is *not* necessarily complete

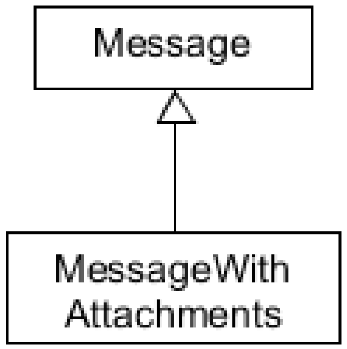


```
String
------------------------------------------
data: char[]
------------------------------------------
contains(s: String): boolean

endsWith(suffix: String): boolean

indexOf(s: String): int

length(): int

replace(target: String,
         replace: String): String

substring(begin: int ,
           end: int): String

toLowerCase(): String
```

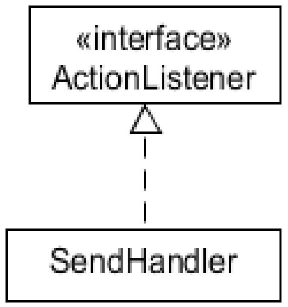
String objects are ***immutable*** – if the method produces a String, the method *returns* that String rather than mutating (changing) the implicit argument

Summary of UML Class Diagram Arrows

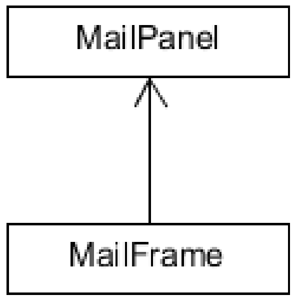
Inheritance (is-a)



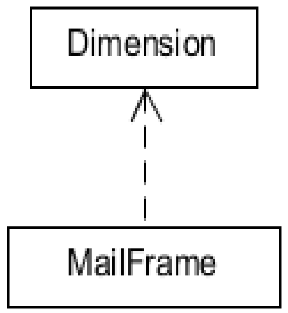
Interface Implementation (is-a)

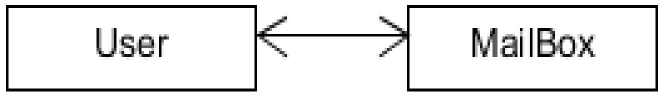


Association (has-a-field)

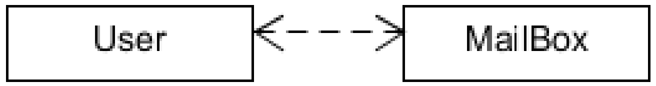


Dependency (depends-on)

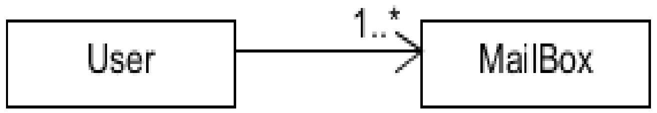




Two-way Association

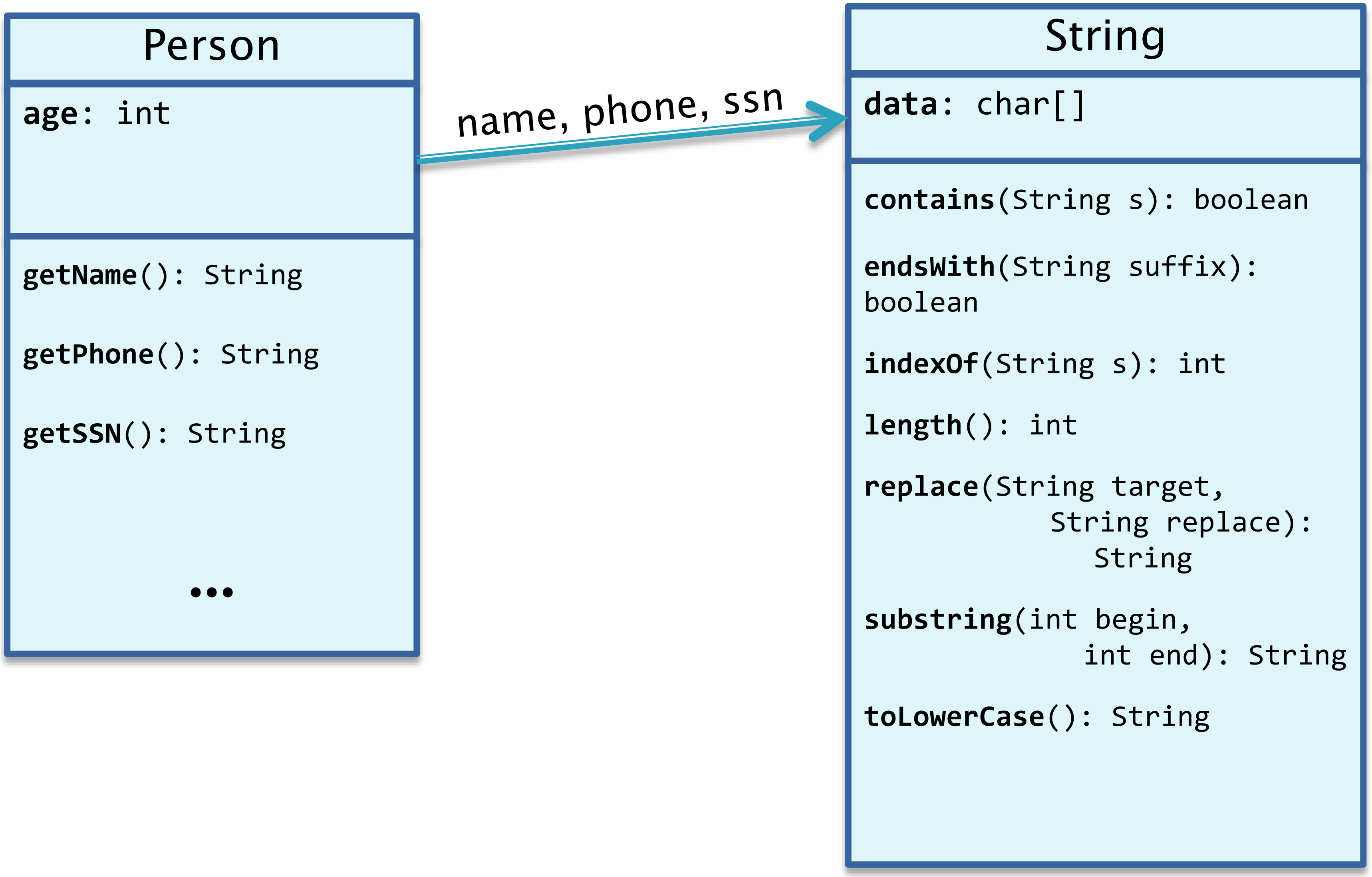


Two-Way Dependency



Cardinality
(one-to-one, one-to-many)
One-to-many is shown on left

Person has String…



3 Things…

- The “things” of what you’re describing usually become the classes
  - The verbs usually become methods of the classes
- Avoid using plurals
  - We make an ArrayList of Face objects, not Faces.
- Make it work!
  - Go through it with some “use case” in mind and make sure that when this object is created, you can accomplish that case. Otherwise, redesign that design until it “works!!!”

Good Classes Typically

- Come from nouns in the problem description
- May…
  - Represent single concepts
    - `Circle`, `Investment`
  - Represent visual elements of the project
    - `FacesComponent`, `UpdateButton`
  - Be abstractions of real-life entities
    - `BankAccount`, `TicTacToeBoard`
  - Be actors
    - `Scanner`, `CircleViewer`
  - Be utility classes that mainly contain static methods
    - `Math, Arrays, Collections`

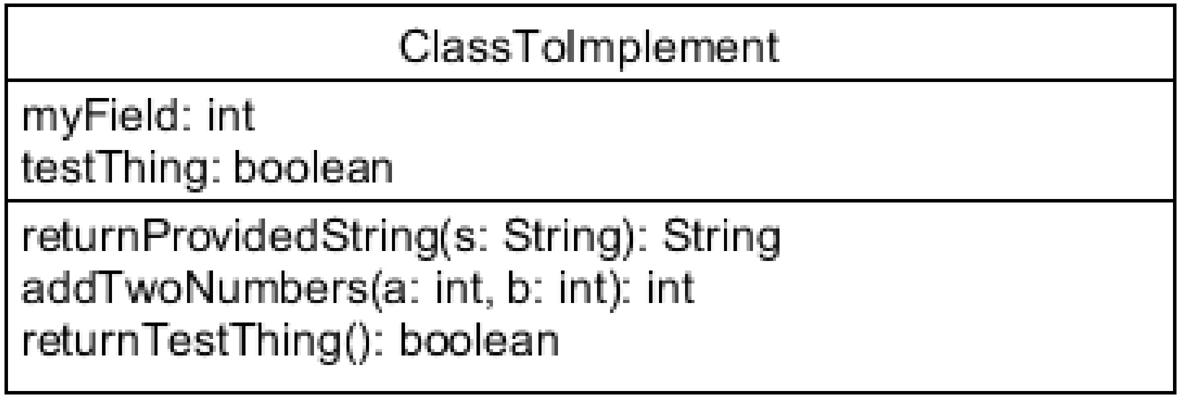
What Stinks? Bad Class Smells*

- Can’t tell what it does from its name
  - `PayCheckProgram`
- Turning a single action into a class
  - `ComputePaycheck`
- Name isn’t a noun
  - `Interpolate`, `Spend`

*See http://en.wikipedia.org/wiki/Code_smell
http://c2.com/xp/CodeSmell.html

Exercises

Complete the questions on the quiz, use the UML shown here for the 3rd question.

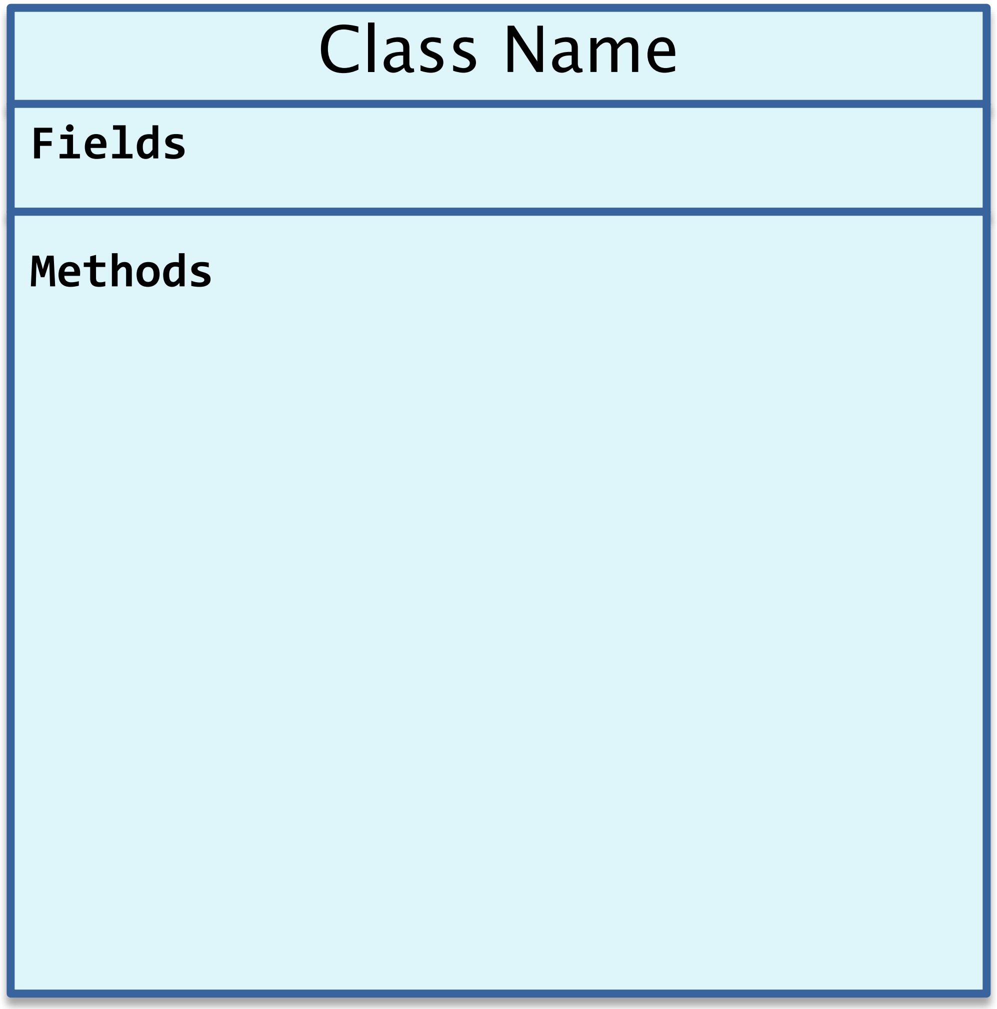

ClassToImplement
myField: int
testThing: boolean
returnProvidedString(s: String): String
addTwoNumbers(a: int, b: int): int
returnTestThing(): boolean

Work with your groups of 3/4

- Decide what classes ought to be in the system and what methods/fields those classes should have (your design should have at least 2 classes)
- Don't forget one class needs to have a main method
- Make sure your design works!
- Write down your answers on a piece of paper with all of your team's names on it
- Call me over when you think you're done – then you'll implement it

Exercise: Class Diagrams

- Task: Make Class diagrams for the Invoice example from OrderTaker

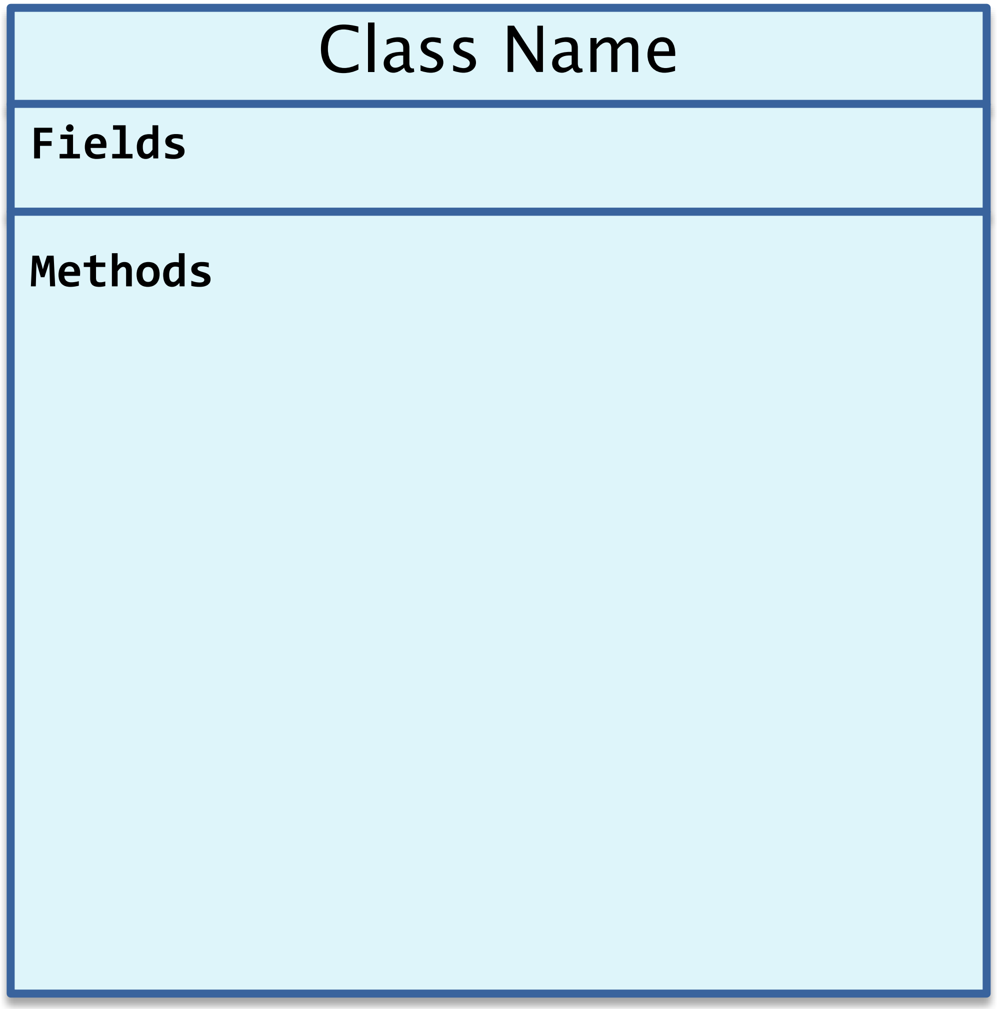


Blackjack – Work with your groups of 4

- Decide what classes ought to be in the system and what methods/fields those classes should have (your design should have at least 3 classes)
- Don't forget one class needs to have a main method
- Make sure your design works!
- Write down your answers on a piece of paper with all of your team's names on it
- Call me over when you think you're done

Exercise: Class Diagrams

- Task: Make Class diagrams for the Simplified Blackjack example
- Make sure all names are on the page. Turn this in for your quiz grade today


Class Name
Fields
Methods