CSE 1710

Lecture 7

Recap and Review of Core Concepts

RQ2.1-2.10

What is a method?

- performs some action
- has a signature and return

range of possibilities?

0 or more parameters, type compatibility must be assured

```
var.methodName( )
Classname.methodName( )
```

What is an attribute?

- holds data
- has a name and a type
- declared and initialized in the class defn

NO parameters

```
var.attributeName
```

In general…

- both are *members* of a class, (also called features)
  - method signatures must be unique, attribute names must be unique
- compiler checks invocations:
  - does the signature (or the attribute name) **match** what is in the class definition?
  - the attributes that clients can access are called fields

RQ2.11

What is scope? What defines scope?

- the term scope is used to refer to the place(s) where it is valid to use a variable
- scope of class attributes
  - valid to use an attribute anywhere (provided that the class has been imported)

RQ2.12

What is a class?

- a definition
- e.g., a description of a car
- gets created in advance
  - it is compiled, is bytecode
- it (the bytecode) gets loaded into runtime memory by the VM upon invocation of an app

What is an object?

- an actual instance of the thing that was defined
- e.g., an actual car
- gets created at runtime
- it gets "born" during runtime, it "dies" during runtime
- has a *state* during runtime
  - specific values for all attributes

RQ2.13-2.14

What do classes have?

- can have *definitions* of:
  - static methods
  - non-static methods
  - static attributes
  - non-static attributes

What do objects have?

- all objects of a given type have the same set of attributes:
  - the static attributes, if any, are common to all objects of a given type; the value must be the same
  - the non-static attributes are specific to each object; the values may differ

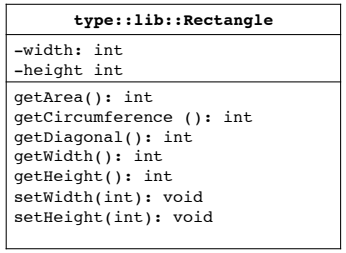
RQ2.15-2.17

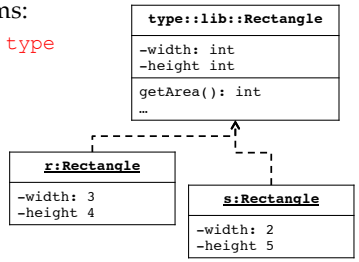
UML Class and Object Diagrams

- full class names separated by colons
  - In java code, full class names separated by dots
- attributes in class diagrams:

  `attributeName : type`
  - a + or – symbol in front means private or public, resp'y
- methods in class diagrams:

  `methodName (param) : type`





RQ2.18

What is abstraction?

- a process whereby details are replaced with something simpler
  - nature of these details?
    - object properties?
      - » abstraction by parameterization
    - details about how a task is performed?
      - » abstraction by delegation

Why do we use abstraction?

- To reduce complexity

RQ2.19

What is an app vs an application?

- app: a class with a main method
- application: an app plus several components

RQ2.20,2.21

What does it mean to be a client?

- to know how & where to look for components
  - understand package structure
  - understand class names may not be unique
  - understand how to read an API, UML diagrams
    - API: a document that specifies what a component does
- to know what you want your app to do
  - not necessarily *how* to implement each and every sub-component
  - you can delegate this to other components
- to know how to use components
  - how to construct objects or otherwise get references to them
    - for delegation of representation, delegation of tasks
  - how to invoke methods, make use of fields
    - static and non-static variants

RQ2.22-2.23

Illustrations of encapsulation

- knowing how to signal a left turn while driving a car does not break encapsulation
  - knowing how to activate the signal does mean knowing how the signal actually operates
    - e.g., how is signal wired? where is the fuse? what is the wattage of the bulb?.
- encapsulation makes the lives of the client and the implementer easier
  - the client needs not know how the component works
  - the implementer needs not know what is the component used for.

RQ2.24

Can the client and implementer roles be occupied simultaneously?

- Depends on who is looking at the situation
  - with respect to end users
    - the end user is the client
    - the application is an implementer
  - with respect to a particular component (no main method)
    - an app that uses the component is the client
    - the component is the implementer

RQ2.25

What does the VM do when a program crashes or has a bug?

- for crashes
  - VM identifies where the problem occurs in the stack trace
- for bugs
  - VM will not realize that there is a bug, so it cannot possibly flag them
- debugging
  - you (not the VM) need to determine why the program produced an incorrect result
  - may need to trace the entire program

RQ2.26

So what is the difference between a bug and other types of errors?

- a bug
  - depends on some notion of what correct output looks like
- compile-time error
  - compiler has a problem with the syntax
  - need to understand compiler's error message
- run-time error
  - VM had a problem running the byte code
  - need to understand stack trace

RQ2.27-2.29

Who's to blame when run-time errors occur?

- run-time error in the main class
  - could be the user
    - provided invalid input?
  - could be the main class
    - has faulty implementation?
- run-time error in a component
  - could be the main class
    - passed invalid parameters?
  - could be the component
    - has faulty implementation?

RQ2.30-2.31

What are the key concepts about Software Engineering?

- it is study of software projects and their progress
- "Risk Mitigation by Early Exposure" is a key principle it is not about program
  - for instance
    - converting the type of a value at runtime is risky
      - e.g., converting a double to an int will result in data loss)
    - the compiler mitigates this risk by checking type compatibility and refuse to compile if there is a violation

RQ2.32

What do I need to know about constants?

- literals embedded in expressions or as parameters are magic numbers
  - you used a literal because:
    - some particular value is needed
    - that particular value is pre-defined and unchanging
- magic numbers should be avoided
- use variables instead of magic numbers
- how do you enforce that the value is predefined and not able to change?
  - use the keyword **`final`** before the declaration.

RQ2.33-2.35

What do I need to know about contracts?

- useful during development and testing
  - stipulates the division of responsibilities:
    - the client
      - needs to ensure the precondition is met
    - the implementer
      - needs to ensure the postcondition is met
  - if precondition is not met, then it is client's responsibility for whatever happens
    - this absolves the implementer of any responsibility
      - implementer may (1) cause crash or (2) return something which may or may not be as specified under the post
    - a dangerous condition can arise if the false precondition does not cause the program to crash