CSE 1710

Lecture 14

Recap and Review of Core Concepts

RQ3.22

How should the program react if an input is invalid?

possible reactions:

1. exception-based approach
2. message-based approach
3. friendly approach

- The key thing is that there *should be* a reaction.
- Invalid input must not be allowed to propagate through.
  - this may result in a logic error that may go undetected (a bad situation!)

RQ2.12

What is a class?

- a definition
- e.g., a description of what a car consists of
- 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., the representation of a particular car
- gets created at runtime
- it gets "born" during runtime, it "dies" during runtime, has a *state* throughout runtime

RQ2.12, pre4.24, pre4.27

What is state?

- an object has state
  - primitive values do not have state, classes do not have state, only objects have state
- objects have zero or more attributes
- attributes hold data
  - data may be primitive or non-primitive
  - contrast with *methods*, which do computation
    - methods may also serve to change the values of the object's attributes
      - » term for such methods?
    - methods may also serve to provide information about the values of the object's attributes
      - » term for such methods?
- "an object's state" refers to **the specific values for all attributes of that object**

What is state? A `Pixel` object example

- Examine the API for `Pixel`
  - any public attributes?
  - can we make any inferences?
  - obtain a reference to a particular pixel object – probe its state using methods (`L14App1`)
- What are the "legal" values for the attributes?
  - has x, y position, must be within bounds of image
  - has R, G, B components, must be in range [0, 255]
  - alpha component must be in range [0, 255]
  - *has an image to which it belongs*
- What is an illegal state for a `Pixel` object?
  - if any component is out of range

Accessors

- What is an accessor?
  - returns info about the state of an object
- Naming convention?
  - if return is boolean: `isXXX()`
  - if return is not boolean: `getXXX()` `getRed()`, …

RQ4.24, 4.26

Accessors

- Is `getAverage()` an accessor?
  - we know the average component value is a secondary value
    - derived on the basis of the three component values
  - the value might be:
    - computed each time we invoke the method
    - represented using a variable, computed each time any of the components change
  - we, as the client, do not know and should not know
    - if we did, this would be breaking the encapsulation

RQ4.25

Accessors

- Can the return of an accessor be `void`?
  - this would be incompatible with the definition of accessor
  - aside – how is `void` different from `null`? `null` is a value; it is valid to assign this value to any object reference. `void` is one of the possible *returns* of a method. It is used to describe the behaviour of a method (namely, it does not return a value of any type). If a method has a non-void non-primitive return, then perhaps the method may return `null` as the value of its return, but this really depends on the contract of the method. (e.g., in the class Pixel, `getColor()` returns a value of type `Color`, but this method should never return `null`.)

RQ4.27

Mutators

- What are some examples of mutators from `Pixel`?
  - `setRed(int)`, `setBlue(int)`, `setGreen(int)`
  - `setColor(Color)`
- What is a mutator in general?
  - provides a means to change the state of an object
  - *more correct:*
    - *provides a means to possibly change the state of an object*

preRQ4.29

Does the mutator always mutate an object?

- what if the parameter value does not make sense?
  - `thePixel.setRed(-50);`
  - `thePixel.setRed(256);`
- look to the API
  - just because there is a contract doesn't mean it's a good one
- mutators enforce the condition that the object not be allowed to go into an illegal state
- why isn't there a mutator for x, y position?
  - the class does not provide services for changing them

RQ4.28

Mutators

- What is the return of `Pixel`'s mutators?
  - void
- In principle, what could be another type of return for a mutator?
  - `boolean`
  - and what would this indicate?
    - indicates whether the requested mutatation was successful

RQ4.29

Mutators and *illegal* states

- How does a mutator enforce that the object is always in a legal state?
  - it enforces conditions prior to any change to the object state

RQ4.30

Why are there no public attributes in Pixel?

- it has good design:
  1. the client can only change object state through mutators
     - the mutators can enforce conditions
  2. the data type(s) and variable names are encapsulated (know "what" not "how")
     - knowing **that** the colour components are represented ("what")
     - vs knowing **how** the colour components are is represented (e.g., as three separate int variables vs an array vs a vector etc) ("how")
- if attributes were public… (both of these things are undesirable)
  - any client could come along and change them, and the class could not impose any conditions
  - clients would know about the "how" – they only need to know about the "what"

RQ4.31

What if the mutators don't enforce any conditions? Can/should the attributes be public in that case?

- No, not even in this case!
- the implementer needs to *encapsulate*
  - implementer may want to add conditions later
  - implementer may want to change way the object state is represented (variable names and/or types)

RQ4.22, 4.23, 4.21

How can we delete the `Pixel` object?

- cannot delete directly
  - there is no method or keyword to be invoked
- possible techniques
  - set reference to `null`
  - this will result in object deletion by the Garbage Collector (but only if there are no other references to the object)
- What do we call an object that has no references to it?
  - an orphan

RQ4.17-4.20

How does Pixel's toString() method behave?

- returns a string like this:
  `Pixel red=60 green=87 blue=34`
- Purpose?
  - provide a textual description
- Is `toString()` obligatory?
  - yes, all classes must have it
- Example of implicit invocation?
  - in `println(…)` (also as operand in string concatenation)
- Default behaviour?
  - returns the class of the object and its location in memory (as a hexadecimal number) (e.g., `PrintStream` has the default implementation)

RQ4.15,4.16

How does Pixel's `equals()` method behave?

- Look at API – we see that the method is inherited from Object
  - Consider `L14App2`
- How does the Java-added method determine equality?
  - it asks the question: are these two references referring to the same object in memory?
  - contrast with the question: are these two references referring to objects that have the same state?
- Is the equals method obligatory?
  - yes, all classes must have it

RQ4.14

Since two `Pixel` objects were deemed equal using `equals`, are they also equal using ==?

- in this case, yes, because:
  - we know equals is using the default version of equals, which is based on ==
- in general, not necessarily, because:
  - the test == is true only if the two objects are the same
  - a class may define the method equals to return true if two objects are different but have the same state

RQ4.13

What about the converse:

If two were deemed equal using ==, are they also equal using `equals`?

- yes, always
- a class can implement equals in whichever way makes sense to it, with one caveat:
  - the identity property must be preserved
  - an object must always be equal to itself

RQ4.4, 4.8

`L14App3` – Discussion

- two different `Pixel` objects
  - either can be mutated and have effect on the Picture
- Walk-through the four steps involved in creating an instance of a `Pixel` object [line
  - locate the class (in memory)
    - it was loaded by the class loader
  - declare the reference
  - instantiate the class
    - the copying model – each object has a set of attributes that get copied
  - assign the reference
- What if the fourth step were omitted?
  - object will not have a reference to it; it will be orphaned

RQ4.4, 4.8

If `Pixel` had a static attribute, how would it this be handled at run-time?

- in step 3, the static attribute would not be copied to the object

RQ4.11

Why could we say that `thePixel2` is an address of an address?

- `thePixel2` (or any variable) is merely a location in memory that holds a value
- in the case of `thePixel2`, the value it holds is another location
- the variable is an address in memory, and that address holds another address