pqChap11

Practice Quiz Chapter 11

[A ] True/False

(a) The tolerant approach to precondition checking is recommended over the demanding approach.(F)

(b) Class invariants are strictly not needed. (T)

(c) Use of functions is allowed when stating assertions. (T)

(d) Correctness is a relative term. (T)

(e) Assertions are Boolean expressions with a few extensions. (T)

(f) Individual components of assertions can be named. (T)

(g) Redundant checks increase the complexity of software. (T)

(h) Complexity is the major enemy of reliability. (T)

(i) Defensive Programming increases the complexity of software. (T)

(j) Design-by-Contract goes against Defensive Programming. (T)

(k) In Eiffel loops, the ‘variant’ and ‘invariant’ parts are optional. (T)

(l) The expression in the (loop) variant must decrease after each iteration of the loop. (T)

(m) The expression in the (loop) variant must never become negative during the

execution of the loop. (T)

(n) ) The main use of a (loop) variant is to show that the loop terminates. (T)

[B] Fill-in:

(a) The Hoare triple for the best job in the world is: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

{false} A {…}

[b] The Hoare triple for the second best job in the world is: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

{…} A {true}

[c] A precondition violation, is a manifestation of a bug in the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

(client)

[d] A postcondition violation, is a manifestation of a bug in the \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

(supplier)

[e] In the “demanding” approach to precondition checking, the precondition is checked by the

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. (client)

[f] In the “tolerant” approach to precondition checking, the precondition is checked by the

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. (supplier)

[g] The level of assertion checking must be specified as a compiler option using a

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ language like Lace, in an Ace file. (control)

(h) The keyword \_\_\_\_\_\_\_\_\_\_\_\_ is used to introduce the pre-condition.

(require)

(i) The keyword \_\_\_\_\_\_\_\_\_\_\_\_\_\_ is used to introduce the post-condition.

(ensure)

(j) The Non-Redundancy principle of Design-by-Contract states that:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

(Under no circumstances shall the body of a routine ever test for the routine’s precondition)

(k) State the Paradox of Software Reliability: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.

(To get more reliability, the best policy is often to check less)

(l) \_\_\_\_\_\_\_\_\_\_\_ are due to defects, which result from \_\_\_\_\_\_\_\_\_\_\_\_\_.

(faults, errors) (see page 347)

[C] Multiple Choice

[1] Pre-conditions are a(n) \_\_\_\_\_\_\_\_\_\_\_\_ for the client. (b)

(a) benefit (b) obligation

[2] Post-conditions are a(n) \_\_\_\_\_\_\_\_\_\_\_\_\_ for the client. (a)

(a) benefit (b) obligation

[3] Pre-conditions are a(n) \_\_\_\_\_\_\_\_\_\_\_\_ for the supplier. (a)

(a) benefit (b) obligation

[4] Post-conditions are a(n) \_\_\_\_\_\_\_\_\_\_\_\_\_ for the supplier. (b)

(a) benefit (b) obligation

[5] Design-by-Contract prefers the \_\_\_\_\_\_\_\_\_\_\_\_\_\_ approach to pre-condition checking. (a)

(a) demanding (b) tolerant

[D] Short Answer

[1] Describe the Imperative-Applicative Opposition (p. 353)

Imperative Applicative

Implementation Specification

Instruction Expression

How What

Imperative Applicative

Prescription Description

[2] State the Reasonable Precondition Principle (p. 356)

Every routine precondition must satisfy the following requirements:

The precondition appears in the official documentation distributed to authors of client modules.

It is possible to justify the need for the precondition in terms of the specification only.

[3] State the Precondition Availability Rule (p. 358)

Every feature appearing in the precondition of a routine must be available to every client to

which the routine is available.

[4] State the Invariant Rule. (p. 366)

An assertion *I* is a correct class invariant for a class *C* if and only if it meets

the following two conditions:

E1 • Every creation procedure of *C*, when applied to arguments satisfying

its precondition in a state where the attributes have their default

values, yields a state satisfying *I*.

E2 • Every exported routine of the class, when applied to arguments and a

state satisfying both *I* and the routine’s precondition, yields a state

satisfying *I*.

[5] Besides pre-conditions, post-conditions, and class invariants, where can assertions be used in Eiffel ?

In ‘check’ constraints, and in (loop) invariants.

[6] What are the different assertion checking levels ?

no, require, ensure, invariant, loop, check.

Excluding ‘no’, each of these levels implies the previous ones.

[7] How much assertion monitoring is needed ?

Generally, during debugging , highest level of assertion checking is recommended.

If you have a fully trusted system (but we can never have one without proofs) you may consider

all monitoring during production use.

An interesting possibility is :

assertion (require)

which checks only preconditions (even for perfect library routines this is recommended).

The purpose is to find errors in client software.

Perform extensive testing with all the checks enabled.

[8] What are the limitations of the Eiffel assertion language ?

It is an engineering compromise. It does not have the expressive power of specification languages like Z, VDM, Larch or OBJ-2. But it tries to overcome limitations by allowing the use of functions.

[9] State the Assertion Evaluation Rule (page 402).

During the process of evaluating an assertion at run time, routine calls shall be executed without any evaluation of the associated assertions.

[10] State two important consequences of the principles of Design by Contract.

(a) A disciplined approach to exception-handling.

(b) How they combine with inheritance.