This is a CLOSED-BOOK quiz.

Before you start, write your name at the top of every sheet.

There are 10 parts (labeled A through J), each with several numbered questions. Please check your copy of the quiz before you start to make sure it is complete: you should have 8 sheets, printed on 13 sides in total.

You have 75 minutes and should attempt to answer all questions. Note that the last 3 questions are worth much more than the multiple choice questions.

<table>
<thead>
<tr>
<th>Part</th>
<th>Score</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>110</td>
<td></td>
</tr>
</tbody>
</table>
A  State Machines

Each numbered part in this section consists of multiple true/false questions. Please answer by writing T or F to the left of each letter label.

1. A non-deterministic state machine model
   a. must have more than one state  **T**
   b. cannot be implemented by a deterministic program  **F**
   c. can be used to model failures  **T**
   d. must have more than one operation  **F**

2. The operations view of a state machine
   a. organizes the transitions by their labels  **T**
   b. does not support non-determinism  **F**
   c. is usually closer to the code than a graphical view  **T**
   d. can accommodate a more complex state than a graphical view  **T**
3. What is the key characteristic of the Singleton pattern? Circle one answer.
   a. at most one caller can access the class at a time
   b. at most one object of the class can be created
   c. at most one class is used to represent a state machine
   d. at most one thread can access a method at a time

4. What is the key characteristic of the State pattern? Circle one answer.
   a. states are represented as methods
   b. states are represented as objects
   c. transitions are represented as classes
   d. transitions are represented as closures

5. What is the key characteristic of the Visitor pattern? Circle one answer.
   a. a structure is visited recursively
   b. the cases of a function are brought together in a single class
   c. an object is visited but never modified
   d. binary functions implemented as static methods

6. Which pattern would be most appropriate for a state machine with multiple, orthogonal state components? Circle one answer.
   a. State
   b. Interpreter
   c. Machine as class
   d. Composite

7. Which pattern would be most appropriate for a state machine with a few gross states, each of which has a set of many substates, each set formed with different types? Circle one answer.
   a. State
   b. Interpreter
   c. Machine as class
   d. Composite
**C  Java Concepts**

8. Which of these best describes the null value? Circle one answer.
   a. an object with no methods
   b. a reference that does not refer to an object
   c. a primitive value
   d. an exception object

9. Which of these best describes overloading in Java? Circle one answer.
   a. replacement of a method in a superclass with a method in a subclass
   b. presence of multiple methods of the same name with different argument types
   c. use of a variable to denote different objects at different times
   d. excessive demand on the processor or memory

10. Which of these elements can be declared in a Java interface? Circle one answer.
    a. static methods
    b. instance variables
    c. constructors
    d. private methods
    e. none of the above

11. Which of these best describes the effect of a type cast? Circle one answer.
    a. the object is converted from one type to another
    b. the type is checked and an exception thrown if incorrect
    c. a new value is created of the appropriate type
    d. a compile-time check alone is produced

12. Which of these best describes the advantage of static typing? Circle one answer.
    a. code is more succinct
    b. fewer errors occur at runtime
    c. exceptions are always caught
    d. rep exposure is prevented
13. Which of these is a reason for adding a type parameter to a class? Circle one answer.
   a. more computations can be expressed
   b. performance is improved
   c. fewer errors occur at runtime
   d. more extensive reuse is possible

14. Why is delegation preferable to inheritance? Circle one answer.
   a. because inheritance violates static typing
   b. because delegation is more efficient
   c. because inheritance damages modularity
   d. because it results in fewer classes

15. Which feature of Java cannot be used to help minimize scope? Circle one answer.
   a. access modifiers
   b. declaration blocks
   c. package namespace
   d. generic types
D  Data abstraction

16. What is the key advantage of immutability? Circle one answer.
   a. simpler, more reliable code
   b. improved performance
   c. better compile–time type checking
   d. less sharing of objects

17. Which values of a datatype need not satisfy the rep invariant? Circle one answer.
   a. values created by a constructor
   b. transient values created internally in methods
   c. values passed to observer methods
   d. values passed to public methods of the datatype by the datatype itself

   a. absence of coupling between representations of two types
   b. the ability to change the representation of a datatype
   c. a representation invariant that is true
   d. the ability to check methods of a datatype one at a time
**Concurrency**

19. What is the most effective way to avoid race conditions? Circle one answer.
   a. program only with immutable types
   b. use concurrency-safe library classes
   c. favor delegation over inheritance
   d. run on a machine with only one processor

20. In which respect is shared-variable concurrency harder than message-passing concurrency? Circle one answer.
   a. shared variables are hard to define in Java
   b. it admits deadlock, which is impossible in message passing
   c. determining what to lock and when is difficult
   d. it requires threads, but message passing does not

21. What is the key distinction between processes and threads? Circle one answer.
   a. threads operate in the same memory space
   b. processes can run on multiple processors, but threads are always confined to a single processor
   c. using processes prevents deadlocks
   d. threads are more lightweight because starting a thread requires less code to be written
F  Design principles

22. Which of the following is *not* a reason for minimizing scope? Circle one answer.
   a. reducing memory usage of a program
   b. localizing the effects of a code modification
   c. making reasoning more modular
   d. avoiding simple cut-and-paste errors

23. Which of the following does *not* contribute to locality? Circle one answer.
   a. using data abstraction
   b. implementing each specification feature in at most one module
   c. preventing representation exposure
   d. using unchecked exceptions
G Testing

Each numbered part in this section consists of multiple true/false questions. Please answer by writing T or F to the left of each letter label.

Branch coverage of the code:

- a. ensures that all bugs will be found [F]
- b. is more demanding than statement coverage [T]
- c. is not needed if the spec-coverage has been used [F]
- d. corresponds to transition coverage of the control flow graph [T]

24. A test stub:

- a. replicates exactly the behaviour of a missing module [F]
- b. can simulate errors that are hard to produce otherwise [T]
- c. is never required in unit testing [F]
- d. can always be generated automatically [F]

25. Inductive generation of values for testing:

- a. is possible only for types with recursive implementations [F]
- b. uses only constructors and producers [T]
- c. guarantees coverage of the specification [F]
- d. ensures preservation of the rep invariant [F]

26. Strengthening a method’s precondition generally:

- a. results in fewer test cases [T]
- b. makes the method harder to implement [F]
- c. helps writing client code [F]
- d. makes the method easier to reason about [T]
H Drawing a State Machine

27. Consider the kind of self-checkin terminal used by most airlines. The passenger identifies him or herself by typing a name or swiping a card, then selects a destination, has options to choose a seat and request an upgrade, and finally obtains a printed boarding card. Construct a state machine diagram for such a terminal in the following stages:

a. List the actions, and give a one-line informal definition of each

   login - The user either types in a name or swipes in a card
   dest - The user chooses the destination he is traveling to
   upgrade - The user selects a seat upgrade
   print - The terminal prints out the boarding pass

b. List the key failures that might occur

   - user unknown
   - network failure
   - printer failure

c. Draw the diagram

   see attached page
I Design Sketch

28. Consider an application for cataloguing photos that offers filtering in which the display can be restricted to the subset of photos that satisfy a filtering predicate. Predicates are formed from basic predicates on metadata attributes, such as whether the capture time falls between two dates, or whether the ISO speed was a particular value, and compound predicates using the logical connectives and, or and not. Construct a design sketch of some code for representing predicates in the following stages:

a. Write some recursive type definitions for the predicates

   \[
   \text{Pred} = \text{Range} (\text{Date}, \text{Date}) \cup \text{Speed} (\text{int}) \cup \text{And} (\text{Pred}, \text{Pred}) \cup \text{Or} (\text{Pred}, \text{Pred}) \cup \text{Not} (\text{Pred})
   \]

b. Draw a type hierarchy showing a box for class and interface, and arrows indicating “extends” and “implements”.

   See last page

c. Write code for each class, giving just the declarations of fields.

   interface Pred { }
   class Range implements Pred { Date start; Date end; }
   class Speed implements Pred { int speed; }
   class And implements Pred { Pred pred1; Pred pred2; }
   class Or implements Pred { Pred pred1; Pred pred2; }
   class Not implements Pred { Pred pred; }
Critiquing Code

29. The code on the next page violates at least four principles of software design and implementation. Find four flaws in the code that violate different principles, and say briefly what each principle is. Do not include the omission of documentation as a flaw.

Line number: 3
Defect: List elements is not declared private
Principle: Reduce scope wherever possible, and do not expose ADT representation; clients or other classes do not need access to elements, and furthermore should not see (and potentially depend on) BadListSet's representation.

Line number: 4
Defect: boolean first is declared static
Principle: Reduce scope wherever possible; first does not need to be shared among all instances of BadListSet and should not be static. This shared state also introduces a concurrency bug.

Line number: 9
Defect: this is checked for reference equality to null
Principle: The this reference in Java can never be null. Additionally, if a check of e.g. (elements == null) had been the programmer's intention, the principle of avoiding null values would apply.

Line number: 14
Defect: Calls to remove(...) enter an infinite recursion
Principle: When implementing a recursive method, we must (a) check for a base case before making a recursive call, and (b) ensure that recursive calls eventually reach the base case. remove(...) enters a recursion that never reaches the base condition it checks for in line 13.

Line number: n/a
Defect: No representation invariant check
Principle: Define representation invariants, make them clear in the code, and use e.g. a checkRep() method to check the RI of a class and expose bugs as early as possible.
import immutable.List;

public class BadListSet<E> implements Set<E> {
  List<E> elements;
  static boolean first;

  public BadListSet (List<E> es) {
    elements = es;
  }

  public Set<E> add (E e) {
    if (this == null) return new BadListSet<E>().add(e);
    else return new BadListSet<E>(elements.add(e));
  }

  public Set<E> remove (E e) {
    if (!contains(e)) return this;
    E x = choose(); Set<E> s = remove(x);
    if (x.equals(e)) return s;
    else return s.remove(e).add(x);
  }

  public boolean contains (E e) {
    return elements.contains(e);
  }

  public E choose () {
    return elements.first();
  }

  public int size () {
    return elements.size();
  }

  public String toString () {
    String result = "Set {";
    Set<E> s = this;
    first = true;
    while (s.size() != 0) {
      if (!first) result += ", "; first = false;
      E e = s.choose(); s = s.remove(e);
      result = result + e;
    }
    return result + "}";
  }
}
Solution to part I, (b)
Solution to part H, (c)