National University of Singapore

SCHOOL OF COMPUTING

CS2030 — PROGRAMMING METHODOLOGY II
(Semester 2: AY2024/2025)

Apr / May 2025 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This assessment contains **TWENTY-TWO(22)** questions.
2. Answer **ALL** questions. The maximum mark is **40**.
3. This is an **EXAMPLIFY-SECURE OPEN-BOOK** assessment. You may refer to your lecture notes, recitation guides, lab programs, and the Java API.
4. You will be liable for disciplinary action which may result in expulsion from the University if you are found to have contravened any of the clauses below,
   - Violation of the NUS Code of Student Conduct (in particular the part on Academic, Professional and Personal Integrity), NUS IT Acceptable Use Policy or NUS Examination rules.
   - Possession of unauthorized materials/electronic devices.
   - Bringing your mobile phone or any storage/communication device with you to the washroom.
   - Unauthorized communication e.g. with another student.
   - Reproduction of any exam materials outside of the exam venue.
   - Photography or videography within the exam venue.
   - Plagiarism, giving or receiving unauthorised assistance in academic work, or other forms of academic dishonesty.
5. Once you have completed the assessment,
   - Click on the "Exam Controls" button and choose "Submit Exam".
   - Check off "I am ready to exit my exam" and click on "Exit" to upload your answers to the server.
   - You will see a green confirmation window on your screen when the upload is successful. Please keep this window on your screen.
   - If you do not see a green window, please disconnect and reconnect your WiFi and try again.
   - Please be reminded that it is your responsibility to ensure that you have uploaded your answers to the Software.

SECTION A (15 Multiple Choice Questions : 15 Marks)
Section A comprises 15 multiple choice questions with five possible answers each. The last option, None of the above, should be chosen if the answer cannot be found in the first four choices. You may assume that all program fragments are syntactically correct and compilable, and that any deviations detected are due to either minor typos or short-cuts and abbreviations used. 1 mark for each correct answer and no penalty for wrong answer.

1. Consider the following Java program fragment.

```
int sum = 1;
for (int x = 1; x <= n; x = x + 1) {
    sum = (sum * x) + 2;
}
return sum;
```

Which of the following is (or are) its equivalent stream-based solution(s)?

i.
```
return IntStream.rangeClosed(1, n)
    .reduce((s, x) -> (s * x) + 2)
    .orElse(1);
```

ii.
```
return IntStream.rangeClosed(1, n)
    .reduce(1, (s, x) -> (s * x) + 2);
```

iii.
```
return IntStream.rangeClosed(1, n)
    .sequential()
    .reduce(1, (s, x) -> (s * x) + 2);
```

iv.
```
return IntStream.rangeClosed(1, n)
    .parallel()
    .reduce(1, (s, x)-> (s * x) + 2);
```

(A) i and ii

(B) ii and iii

(C) i, ii, and iii

(D) i, ii, iii and iv

(E) None of the above.

2. Consider the following program fragment:

```
str.map(f).reduce(I,cmb)
```

where

- `str` is stream of type `Stream<T>`;
- `f` is a pure function of type `Function<T,R>`;
- `cmb` is a pure function of type `BinaryOperator<R>` satisfying the associative property;
- `I` is the identity of `cmb`.

Which of the following is/are its equivalent the above program fragment?

i. `str.reduce((r, x) -> cmb.apply(r, f.apply(x)))`

ii. `str.reduce(I, (r, x) -> cmb.apply(r, f.apply(x)))`

iii. `str.reduce(I, (r, x) -> cmb.apply(r, f.apply(x)), cmb)`

iv. `str.parallel().reduce(I, (r, x) -> cmb.apply(r, f.apply(x)), cmb)`

(A) i and ii

(B) ii and iii

(C) ii, iii, and iv

(D) i, ii, iii and iv

(E) None of the above.

3. Let us compare Java's *interface* mechanism against the *abstract class* mechanism. Which of the following statements is NOT true?

   (A) Every method of each Java interface must be public, while those of abstract classes could be private.

   (B) Every data field of interfaces must be final and static, while those of abstract classes maybe non-final and/or non-static.

   (C) Each concrete class is allowed to implement multiple interfaces, but may only extend at most one abstract class.

   (D) Every method in an interface must always be *abstract* (without implementation), while each method in abstract class may be either *abstract* or *concrete* (with implementation).

   (E) None of the above.

4. Consider the following static method declaration:

```
static <R,T> R foo(Stream<T> str, R I,
    Function<T,R> f, BinaryOperator<R> cmb) {
    Stream<R> tmp = str.map(f);
    return tmp.reduce(I, cmb);
}
```

Which of the following is the best wildcard generic parameters' types that you can give for the above `foo` method.

(A)
```
static <R,T> R foo(Stream<T> str, R I,
        Function<T,R> f,
        BinaryOperator<R> cmb)
```

(B)
```
static <R,T> R foo(Stream<? extends T> str, R I,
        Function<T,R> f,
        BinaryOperator<R> cmb)
```

(C)
```
static <R,T> R foo(Stream<? extends T> str, R I,
        Function<? super T,? extends R> f,
        BinaryOperator<R> cmb)
```

(D)
```
static <R,T> R foo(Stream<? extends T> str, R I,
        Function<? super T,? extends R> f,
        BinaryOperator<? extends R> cmb)
```

(E) None of the above.

5. Consider the `infPrimes` methods defined in JShell to compute an infinite stream of prime numbers :

```
IntStream infPrimes() {
    return IntStream.iterate(2,x->x+1).filter(x -> isPrime(x));
}

boolean isPrime(int n) {
    return n > 1 && IntStream.range(2, n)
        .noneMatch(x -> n % x == 0);
}
```

You are asked to compute the number of prime numbers between `n` and `m` inclusive, where `2 <= n <= m`. For example, if `n = 10` and `m = 19`, you would need to count the following prime numbers `List(11, 13, 17, 19)` and thus return `4`. Which of the following call invocations when applied to `infPrimes()` would perform such a count correctly?

i.
```
infPrimes().limit(m - n + 1).
      filter(x -> x >= n && x <= m).
      count()
```

ii.
```
infPrimes().filter(x -> x >= n && x <= m).
      limit(m - n + 1).
      count()
```

iii.
```
infPrimes().limit(m).
      filter(x -> x >= n && x <= m).
      count()
```

iv.
```
infPrimes().limit(m).
      filter(x -> x >= n && x <= m).
      map(x -> 1).
      sum()
```

v.
```
infPrimes().limit(m).
      filter(x -> x >= n && x <= m).
      reduce(0, (x, y) -> x + 1)
```

vi.
```
infPrimes().limit(m).
      filter(x -> x >= n && x <= m).
      reduce(0, (x, y) -> 1 + y)
```

(A) i, ii and iii

(B) ii, iii and iv

(C) iii, iv, and v

(D) iv, v and vi

(E) None of the above.

6. Consider two class declarations below with four methods of the same name.

```
class A {
   public t1 mn(A v) {
       ...
   }

   public t2 mn(Object v) {
       ...
   }
}

class B extends A {
   public t3 mn(B v) {
       ...
   }

   public t4 mn(Object v) {
       ...
   }
}
```

For these methods to compile successfully, which of the following type constraint(s) must hold. Note that `t1 <: t2` denotes that type `t1` is a subtype of type `t2`.

i. `t1 <: t3`

ii. `t2 <: t4`

iii. `t3 <: t1`

iv. `t4 <: t2`

(A) i

(B) ii

(C) ii and iv

(D) iii and iv

(E) None of the above.

7. Consider the following program fragment:

```
class A {
   public void foo(A v) {
       System.out.print("1");
   }

   public void foo(Object v) {
       System.out.print("2");
   }
}

class B extends A {
    public void foo(A v) {
        System.out.print("3");
    }
    public void foo(B v) {
        System.out.print("4");
    }
}

B b = new B()
A a1 = b
A a2 = new A()
Object o = a2
b.foo(b)
a1.foo(a1)
a2.foo(a2)
a1.foo(o)
```

What output will be printed when the above program fragment is executed?

(A) `4332`

(B) `4321`

(C) `4312`

(D) `4132`

(E) None of the above.

8. Consider the following `List` object.

```
List<String> list = List.of("a","b","c")
```

Which of the following methods can be executed without any errors (at both runtime and compile-time)? Consider the following program fragment:

i. `list.add("z")`

ii. `list.get(2)`

iii. `list.size()`

iv. `list.contains("z")`

v. `list.clear()`

(A) i and iv

(B) ii and iii

(C) ii, iii, and iv

(D) i, ii, iii, iv and v

(E) None of the above.

9. Consider the following program fragment:

```
interface I<T,R> {
    R goo(T x);
}

static .. hoo(.. a, .. b) {
    return a.goo(b);
}
```

Which of the following is the *most general* Java type declaration for the `hoo` method?

(A) `static <A extends I<A,R> ,R> R hoo(A a, A b)`

(B) `static <A extends I<? super A, ? extends R>,R> R hoo(A a, R b)`

(C) `static <A extends I<? super B, ? extends R>,B,R> R hoo(A a, B b)`

(D) `static <A extends I<? extends B, ? extends R>,B,R> R hoo(A a, B b)`

(E) None of the above.

10. Consider the following program fragment:

```
try {
    System.out.print("1");
    throw new Exception("e");
} catch (RuntimeException e) {
    System.out.print("2");
    throw new Exception("f");
} finally {
    System.out.print("3");
}
```

What is its expected outcome when the above is executed?

(A) outputs `12`

(B) outputs `12`, followed by `Exception("f")`

(C) outputs `123`, followed by `Exception("f")`

(D) outputs `13`, followed by `Exception("e")`

(E) None of the above.

11. Consider the API for a Lazy context (with caching).

```
class Lazy<T> {
    static <E> Lazy<E> of(E x) { .. }

    static <E> Lazy<E> of(Supplier<E> f) { .. }

    <R> Lazy<R> map(Function<T,R> f) { .. }

    <R> Lazy<R> flatMap(Function<T,Lazy<R>> f) { .. }

    T get() { .. }
}
```

What output will be printed by JShell for the following program fragment?

```
Lazy<Integer> v1 = Lazy.of(() -> {
        System.out.print("Eval 1;");
        return 1; })

Lazy<Integer> v2 = Lazy.of(() -> {
        System.out.print("Eval 2;");
        return 2; })

Lazy<Integer> v3 = v1.map(x -> x + 2)

Lazy<Integer> v4 = v1.flatMap(x ->
        Lazy.of(() -> {
            System.out.print("Eval 4;");
            return x + 4; }))

List<Lazy<Integer>> xs = List.of(v1,v2,v3,v4)

System.out.print(xs.size());
System.out.print(xs.get(2).get());
System.out.print(xs.get(3).get());
System.out.print(xs.get(0).get());
System.out.print(xs.get(1).get());
```

(A) `Eval 1;Eval 4;4351Eval 2;2`

(B) `Eval 1;43Eval 4;51Eval 2;2`

(C) `4Eval 1;3Eval 4;51Eval 2;2`

(D) `4Eval 1;3Eval 1;Eval 4;51Eval 2;2`

(E) None of the above.

12. Consider the following Java class declaration.

```
class Counter {
    private final int a;
    private int b;

    Counter(int a,int b) {
        this.a = a;
        this.b = b;
    }

    void dec() {
        this.b = this.b - 1;
    }

    Predicate<Integer> proc1(int y) {
        return x -> x >= y;
    }

    Predicate<Integer> proc2(int y) {
        y = y + 1;
        return x -> x >= y;
    }
    Predicate<Integer> proc3() {
        return x -> x >= this.a;
    }
    Predicate<Integer> proc4() {
        return x -> x >= this.b;
    }
}
```

Which of these four methods will encounter a closure compilation error *local variables referenced from a lambda expression must be final or effectively final*?

i. `Proc1`

ii. `Proc2`

iii. `Proc3`

iv. `Proc4`

(A) ii

(B) ii and iv

(C) ii, iii, iv

(D) i, ii, iii and iv

(E) None of the above.

13. Given a simplified API for CompletableFuture (abbreviated as CF) shown below.

```
class CF<T> implements Future<T>,CF<T> {
    static <U> CF<U> supplyAsync(Supplier<U> s) { .. }

    <U> CF<U> thenApply(Function<T,U> f) { .. }

    <U> CF<U> thenComposeAsync(Function<T,CF<U>> f) { .. }

    <U,V> CF<V> thenCombine<CF<U> u, BiFunction<T,U,V> f) {...}

    T join() { .. } // returns result when completed or an exception
}
```

Consider the following code fragment where each `delay()` call incurs a finite but non-deterministic delay.

```
v1 = CF.supplyAsync(() -> { delay(); System.out.print("1");
        return 1; })
v2 = CF.supplyAsync(() -> { delay(); System.out.print("2");
        return 2; })
v3 = CF.supplyAsync(() -> { delay(); System.out.print("3");
        return 3; })
v4 = v3.thenComposeAsync(x -> {delay(); System.out.print("4");
        return 4; })
v5 = v1.thenApply(x -> { x2 = v2.join();
        delay(); System.out.print("5");
        return 5; })
v6 = v4.thenCombine(v5, (x,y) -> { delay(); System.out.print("6");
        return 6; })
```

Which of the following output is NOT possible?

(A) `123456`

(B) `125346`

(C) `231546`

(D) `324156`

(E) None of the above.

14. Continuing from question 13 with the code fragment reproduced below:

```
class CF<T> implements Future<T>,CF<T> {
    static <U> CF<U> supplyAsync(Supplier<U> s) { .. }

    <U> CF<U> thenApply(Function<T,U> f) { .. }

    <U> CF<U> thenComposeAsync(Function<T,CF<U>> f) { .. }

    <U,V> CF<V> thenCombine<CF<U> u, BiFunction<T,U,V> f) {...}

    T join() { .. } // returns result when completed or an exception
}

v1 = CF.supplyAsync(() -> { delay(); System.out.print("1");
        return 1; })
v2 = CF.supplyAsync(() -> { delay(); System.out.print("2");
        return 2; })
v3 = CF.supplyAsync(() -> { delay(); System.out.print("3");
        return 3; })
v4 = v3.thenComposeAsync(x -> {delay(); System.out.print("4");
        return 4; })
v5 = v1.thenApply(x -> { x2 = v2.join();
        delay(); System.out.print("5");
        return 5; })
v6 = v4.thenCombine(v5, (x,y) -> { delay(); System.out.print("6");
        return 6; })
```

Let us now assume that we are running this asynchronous program on a CPU with 16 processors. If every `delay()` takes exactly one second and the rest of the computation takes negligible timing, how many seconds will this asynchronous program require to complete its computation.

(A) `3`

(B) `4`

(C) `5`

(D) `6`

(E) None of the above.

15. Consider an abstract monad class with the `unit` and `flatMap` methods.

```
record Pair<T,U>(T t, U u) {};

abstract class Monad<T> {
    abstract <R> Monad<R> unit(R a);

    abstract <R> Monad<R> flatMap(Function<T,Monad<R>> f);
}
```

You decided to design a new operation for your monad call that can perform a given monad `m` twice, as follows;

```
static <T> Monad<Pair<T,T>> twice(Monad<T> m) {
    Monad<Pair<T,T>> r = m.flatMap(x ->
            m.flatMap(y -> m.unit(new Pair<T,T>(x, y))));
    return r;
}
```

Which of the following statement is NOT true about the `twice` method?

(A) The effect of Monad `m` will be performed exactly twice by this method in strict sequence.

(B) The types of both `x` and `y` are always the same, namely `T`.

(C) The values of both `x` and `y` are always the same.

(D) The method twice is a *pure* function if `m` is pure.

(E) None of the above.

SECTION B (7 Questions : 25 Marks) Unless otherwise specified, do not define your own classes or use classes other than those specified in the Java API, even if they have been defined in the course such as `Pair`, `Lazy`, `Try`, `Maybe`, `Log`, etc. You do not need to write import statements.

Questions 16 to 19 refer to the task described below.

The structured query language (SQL) is a declarative language for managing a relational database that is represented by a table. Examples of SQL sessions are shown below. Assuming a table has been created called `books` with four columns `isbn`, `title`, `author`, `price`

- to insert a new row into the table

  ```
  INSERT INTO books
  VALUES (9780394800011, 'The Cat in the Hat', 'Dr. Seuss', 12.48)
  ```

- to select row(s) within the table

  ```
  SELECT price
  FROM books;
  ```

  The above SQL query selects the `price` column from all rows of the `books` table.

  ```
  SELECT *
  FROM books
  WHERE price > 10.00
  ```

  The above SQL query selects ALL columns from only rows that meet the constraint `price > 10.00` of the `books` table.

- to update row(s) within the table

  ```
  UPDATE books
  SET price = 10.00
  WHERE isbn = 9780394800011
  ```

You are to simulate SQL using Java Streams. The following `Book` record and generic `DB` interface is given to you.

```
record Book(Long isbn, String title, String author, double price) {}

interface DB<T> {
    static <T> Stream<T> create() {
        return Stream.<T>of();
    }
}
```

Take note that the return type of `create` is a `Stream` that represents the table. Moreover, you may assume that streams are never reused. Hence, DO NOT convert stream to a list. For simplicity, there is no need to use bounded wildcards.

16. [**2 marks**] Include a static method `insert` that takes in a table and a row, and inserts the row into the table. A sample run is given below:

```
jshell> Stream<Book> books = DB.<Book>create()
books ==> java.util.stream.ReferencePipeline..

jshell> DB.<Book>insert(books,
   ...> new Book(9780394800011L, "The Cat in the Hat", "Dr. Seuss", 12.48)).
   ...> toList()
$.. ==> [Book[isbn=9780394800011, title=The Cat in the Hat, author=Dr. Seuss,
price=12.48]]
```

ANSWER:

17. [**4 marks**] Include two overloaded static methods `select`:

    - one that takes in a table, and the columns of values;
    - one that takes in a table, the column of values, as well as a constraint on the rows to select.

    A sample run is given below:

```
jshell> Stream<Book> books = DB.<Book>create()
books ==> java.util.stream.ReferencePipeline..

jshell> books = DB.<Book>insert(books,
   ...> new Book(9780394800011L, "The Cat in the Hat", "Dr. Seuss", 12.48))
books ==> java.util.stream.ReferencePipeline..

jshell> record Pair<T,U>(T t, U u) {}
|  created record Pair

jshell> DB.select(books, x -> new Pair<Long,Double>(x.isbn(), x.price())).
   ...> toList()
$.. ==> [Pair[t=9780394800011, u=12.48]]

jshell> books = DB.<Book>create()
books ==> java.util.stream.ReferencePipeline..

jshell> books = DB.<Book>insert(books,
   ...> new Book(9780394800011L, "The Cat in the Hat", "Dr. Seuss", 12.48))
books ==> java.util.stream.ReferencePipeline..

jshell> DB.select(books, x -> new Pair<Long,Double>(x.isbn(),
   ...> x.price()), x -> x.price() < 10.00).
   ...> toList()
$.. ==> []
```

    In your answer, write the more *general* `select` method first. Make the other `select` method use the more general version.

    ANSWER:

18. [**3 marks**] Write the static method `update` that returns an updated table based on the values to set and the rows for which the update is to be done. A sample run is given below:

```
jshell> Stream<Book> books = DB.<Book>create()
books ==> java.util.stream.ReferencePipeline..

jshell> books = DB.<Book>insert(books,
   ...> new Book(9780394800011L, "The Cat in the Hat", "Dr. Seuss", 12.48))
books ==> java.util.stream.ReferencePipeline..

jshell> DB.<Book>update(books,
   ...> x -> new Book(x.isbn(), x.title().toUpperCase(), x.author(), x.price()),
   ...> x -> x.price() > 10.0).
   ...> toList()
$.. ==> [Book[isbn=9780394800011, title=THE CAT IN THE HAT, author=Dr. Seuss,
price=12.48]]
```

ANSWER:

19. [**4 marks**] To mimic SQL more closely, write an overloaded static method `update` so that a `where` method can be chained right after `update`. You will need to define both methods. A sample run is given below.

```
jshell> Stream<Book> books = DB.<Book>create()
books ==> java.util.stream.ReferencePipeline..

jshell> books = DB.<Book>insert(books,
   ...> new Book(9780394800011L, "The Cat in the Hat", "Dr. Seuss", 12.48))
books ==> java.util.stream.ReferencePipeline..

jshell> DB.<Book>update(books,
   ...> x -> new Book(x.isbn(), x.title().toUpperCase(), x.author(), x.price())).
   ...> where(x -> x.price() > 10.0).
   ...> toList()
$.. ==> [Book[isbn=9780394800011, title=THE CAT IN THE HAT, author=Dr. Seuss,
price=12.48]]
```

Hint: You may need to define an additional helper class.

ANSWER:

Questions 20 to 22 refer to the task described below.

A `Bag` record is defined below to represent a bag of `u` objects of the same type `T`.

```
record Bag<T>(T t, int u) {}
```

For example, a bag of five red marbles can be represented as

```
jshell> new Bag<String>("red", 5)
$.. ==> Bag[t=red, u=5]
```

In the following questions, you are only allowed to write a single Stream pipeline within each method. DO NOT perform any list processing using loop constructs. For simplicity, there is no need to use bounded wildcards.

20. [**4 marks**] Write a generic method `repack` in JShell that takes a list of `Bags` and repacks the bags such that bags containing the same things are packed into one. A sample run is shown below.

```
jshell> List<Bag<String>> bags = List.of(new Bag<String>("red", 5),
   ...> new Bag<String>("blue", 3), new Bag<String>("red", 2
))
bags ==> [Bag[t=red, u=5], Bag[t=blue, u=3], Bag[t=red, u=2]]

jshell> repack(bags)
$.. ==> [Bag[t=red, u=7], Bag[t=blue, u=3]]
```

The method header type declaration for repack in JShell is as follows:

```
<T> List<Bag<T>> repack(List<Bag<T>> bags)
```

ANSWER:

21. [**4 marks**] Write a generic method `choose` that takes in a list of bags and removes one item from the first bag. The method returns the removed item and the resulting list as a `Pair`. A sample run is shown below.

```
jshell> record Pair<T,U>(T t, U u) {}
|  created record Pair

jshell> List<Bag<String>> bags = List.of(new Bag<String>("red", 5),
   ...> new Bag<String>("blue", 3), new Bag<String>("red", 2)
)
bag ==> [Bag[t=red, u=5], Bag[t=blue, u=3], Bag[t=red, u=2]]

jshell> choose(bags)
$.. ==> Optional[Pair[t=red, u=[Bag[t=red, u=4], Bag[t=blue, u=3],
Bag[t=red, u=2]]]]

jshell> choose(List.<Bag<String>>of())
$.. ==> Optional.empty
```

- Write the header of the `choose` method.
- Complete the body of the `choose` method.

ANSWER:

```
... {
    return list.stream()...
}
```

22. [**4 marks**] Write a generic method `repeat` that takes in a list of bags and a `Consumer<String>` and repeatedly chooses an item to be consumed. A sample run is given below:

```
jshell> List<Bag<String>> bags = List.of(new Bag<String>("red", 5),
   ...> new Bag<String>("blue", 3), new Bag<String>("red", 2)
)
bag ==> [Bag[t=red, u=5], Bag[t=blue, u=3], Bag[t=red, u=2]]

jshell> Consumer<String> consumer = x -> System.out.println(x)
consumer ==> $Lambda..

jshell> repeat(bags, consumer)
red chosen with [Bag[t=red, u=4], Bag[t=blue, u=3], Bag[t=red, u=2]] left
red chosen with [Bag[t=red, u=3], Bag[t=blue, u=3], Bag[t=red, u=2]] left
red chosen with [Bag[t=red, u=2], Bag[t=blue, u=3], Bag[t=red, u=2]] left
red chosen with [Bag[t=red, u=1], Bag[t=blue, u=3], Bag[t=red, u=2]] left
red chosen with [Bag[t=blue, u=3], Bag[t=red, u=2]] left
blue chosen with [Bag[t=blue, u=2], Bag[t=red, u=2]] left
blue chosen with [Bag[t=blue, u=1], Bag[t=red, u=2]] left
blue chosen with [Bag[t=red, u=2]] left
red chosen with [Bag[t=red, u=1]] left
red chosen with none left
```

- Write the header of the `repeat` method.
- Complete the body of the `repeat` method.

ANSWER:

```
... {
    Stream.iterate(choose(list), lst -> lst.isPresent(), ...
}
```

— END OF PAPER —