1

Profs. Martin Odersky and Sanidhya Kashyap
CS-206 Parallelism and Concurrency
27.04.2022 from 14h15 to 15h45
Duration : 90 minutes

SCIPER : 1000001 ROOM : CO1

Ada Lovelace

Wait for the start of the exam before turning to the next page. This document is printed double sided, 4 pages. Do not unstaple.

- This is a closed book exam. No electronic devices allowed.
- Place on your desk: your student ID, writing utensils place all other personal items below your desk or on the side.
- You each have a different exam. For technical reasons, **do use black or blue pens for the MCQ part, no pencils!** Use white corrector if necessary.
- **Your Time:** All points are not equal: we do not think that all exercises have the same difficulty, even if they have the same number of points.
- **Your Attention:** The exam problems are precisely and carefully formulated, some details can be subtle. Pay attention, because if you do not understand a problem, you cannot obtain full points.
- The two last pages of this exam contains an appendix. Do not detach this sheet.

First part: single choice questions

*Each question has **exactly one** correct answer. Marking only the box corresponding to the correct answer will get you 4 points. Otherwise, you will get 0 points for the question.*

Question 1 What are the possible values of the variable `sum` after the execution of the snippet below?

```
var sum = 0
val t1 = task {sum += 1}
val t2 = task {sum += 1}
t1.join()
t2.join()
```

- ☐ {1, 2}
- ☐ {2}
- ☐ {0}
- ☐ {1}
- ☐ {0, 2}
- ☐ {0, 1, 2}

Second part: yes/no questions

*The answer of each question is **either "Yes", either "No"**. Marking only the box corresponding to the correct answer will get you 2 points. Marking only the wrong answer will get you -1 point. Otherwise, you will get 0 point for the question.*

Question 2 Can the following snippet result in a deadlock?

```
class Account(private var amount: Int = 0) extends Monitor:
    def transfer(target: Account, n: Int) =
        this.synchronized {
            target.synchronized {
                this.amount -= n
                target.amount += n
            }
        }

val a = new Account(50)
val b = new Account(70)
val t1 = task { a.transfer(b, 10) }
val t2 = task { b.transfer(a, 10) }
t1.join()
t2.join()
```

☐ Yes ☐ No

Third part, open questions

...

Appendix: Scala and Java Standard Library Methods

Here are the prototypes of some Scala and Java classes that you might find useful:

```
// Represents optional values. Instances of Option are either an instance of
// scala.Some or the object None.
abstract class Option[A]:
  // Returns the option's value if the option is an instance of scala.Some, or
  // throws an exception if the option is None.
  def get: A
  // Returns true if the option is an instance of scala.Some, false otherwise.
  // This is equivalent to:
  //   option match
  //     case Some(v) => true
  //     case None  => false
  def isDefined: Boolean

abstract class Iterable[+A]:
  // Selects all elements except first n ones.
  def drop(n: Int): Iterable[A]
  // Selects all elements of this iterable collection which satisfy a predicate.
  def filter(pred: (A) => Boolean): Iterable[A]
  // Apply f to each element for its side effects.
  def foreach[U](f: (A) => U): Unit
  // The size of this collection.
  def size: Int
  // Selects the first n elements.
  def take(n: Int): Iterable[A]

abstract class List[+A] extends Iterable[A]:
  // Adds an element at the beginning of this list.
  def ::[B >: A](elem: B): List[B]
  // Get the element at the specified index.
  def apply(n: Int): A
  // Selects all elements of this list which satisfy a predicate.
  def filter(pred: (A) => Boolean): List[A]

abstract class Vector[+A] extends Iterable[A]:
  // Get the element at the specified index.
  def apply(n: Int): A
```

```
// An int value that may be updated atomically.
// The constructor takes the initial value at its only argument. For example,
// this create an `AtomicInteger` with an initial value of `42`:
//     val myAtomicInteger = new AtomicInteger(42)
abstract class AtomicInteger:
  // Atomically adds the given value to the current value.
  def addAndGet(delta: Int): Boolean
  // Atomically sets the value to the given updated value if the current value
  // == the expected value. Returns true if the change is successful, or false
  // otherwise. This is an atomic operation.
  def compareAndSet(oldvalue: Int, newvalue: Int): Boolean
  // Gets the current value. This is an atomic operation.
  def get(): Int
  // Atomically increments by one the current value. This is an atomic operation.
  def incrementAndGet(): Int

// A concurrent hash-trie or TrieMap is a concurrent thread-safe lock-free
// implementation of a hash array mapped trie.
abstract class TrieMap[K, V]:
  // Retrieves the value which is associated with the given key. Throws a
  // NoSuchElementException if there is no mapping from the given key to a
  // value.
  def apply(key: K): V
  // Tests whether this map contains a binding for a key.
  def contains(key: K): Boolean
  // Applies a function f to all elements of this concurrent map. This function
  // iterates over a snapshot of the map.
  def foreach: Iterator[K]
  // Optionally returns the value associated with a key.
  def get(key: K): Option[V]
  // Collects all key of this map in an iterable collection. The result is a
  // snapshot of the values at a specific point in time.
  def keys: Iterator[K]
  // Transforms this map by applying a function to every retrieved value. This
  // returns a new map.
  def mapValues[W](f: V => W): TrieMap[K, V]
  // Associates the given key with a given value, unless the key was already
  // associated with some other value. This is an atomic operation.
  def putIfAbsent(k: K, v: V): Option[V]
  // Removes a key from this map, returning the value associated previously with
  // that key as an option.
  def remove(k: K): Option[V]
  // Removes the entry for the specified key if it's currently mapped to the
  // specified value. This is an atomic operation.
  def remove(k: K, v: V): Boolean
  // Replaces the entry for the given key only if it was previously mapped to a
  // given value. Returns true if the change is successful, or false otherwise.
  // This is an atomic operation.
  def replace(k: K, oldvalue: V, newvalue: V): Boolean
  // Adds a new key/value pair to this map.
  def update(k: K, v: V): Unit
  // Collects all values of this map in an iterable collection. The result is a
  // snapshot of the values at a specific point in time.
  def values: Iterator[V]
```