hochschule mannheim

Test 2 (Studienleistung)
PP – Parallele Programmierung
Bachelor Informatik (IB)
Wintersemester 2019/2020 [25.11.2019]

FAKULTÄT FÜR INFORMATIK

Matrikel-Nr.: ..... ☐ CSB / ☐ IB / ☐ IMB / ☐ MEB / ☐ UIB / ☐ IM

Name: .....

Vorname: .....

Unterschrift: .....

Hinweise:

(1) Dieser Test hat 6 Seiten. Sie haben 30 Minuten Zeit zur Beantwortung.

(2) Schreiben Sie Ihren *Namen und Ihre Matrikelnummer zu Beginn* auf das Deckblatt und kreuzen Sie Ihren Studiengang an. Überprüfen Sie, ob der Test vollständig ist.

(3) Schreiben Sie die *Lösung* zu einer Aufgabe *auf das vorgesehene Blatt*. Sollte der Platz nicht reichen, benutzen Sie bitte den Reserveplatz am Ende. Notieren Sie dabei am vorgesehenen Platz, dass es beim Reserveplatz weitergeht, und kennzeichnen Sie auch unbedingt dort, zu welcher Aufgabe die Antwort gehört.

(4) Zerlegen Sie die Heftung *nicht in einzelne Blätter*. Lose Blätter werden nicht gewertet.

(5) Es sind keine Hilfsmittel erlaubt.

(6) Ein Täuschungsversuch führt zur Bewertung mit 0 Punkten. Dazu zählt Kommunikation mit anderen Personen außer Prüfer/in/Aufsichtspersonen und jede Benutzung von unerlaubten Hilfsmitteln wie mobilen Endgeräten (z. B. Smartphones).

(7) Schreiben Sie mit dokumentenechten Stiften (Kugelschreiber). Mit Bleistiften, Füllern o. ä. erstellte Lösungen sind ungültig! Tipp-Ex und rotschreibende Stifte sind ebenfalls verboten. Schreiben Sie bitte leserlich.

Aufgabe	1	2	3	Summe
Punkte	10	10	10	30
Erreicht				

Viel Erfolg!

Aufgabe 1

Aufgabe 1: von 10 P

(a) In der folgenden Java-Klasse soll ein Thread-sicherer Zähler mit `AtomicInteger` realisiert werden, der so effizient wie möglich ist. Korrigieren Sie etwaige Fehler direkt im folgenden Listing: 5 P

```
import java.util.concurrent.atomic.AtomicInteger;
class Counter {
    private AtomicInteger c = new AtomicInteger(0);
    public synchronized void increment() {
        int temp = this.c.get();
        if (!this.c.compareAndSet(temp, +1)) {
            temp = this.c.get();
        }
    }
    public synchronized void decrement() {
        int temp = this.c.get();
        if (!this.c.compareAndSet(temp, -1)) {
            temp = this.c.get();
        }
    }
    public synchronized int value() {
        return this.c.get();
    }
}
```

(b) Kann die Methode `addI` von `Test` unter alleiniger Verwendung von `AtomicInteger` in eine Thread-sichere Form gebracht werden? Falls ja, geben Sie rechts neben dem Listing den Quelltext der geänderten Methode an. Falls es nicht möglich sein sollte `addI` von `Test` unter alleiniger Verwendung von `AtomicInteger` in eine Thread-sichere Form zu bringen, nennen Sie bitte den Grund dafür. 5 P

```
class Test {
    private int i;
    private int j;
    public void addI(int x) {
        i = i + (j * 3) + x;
    }
    /* ... weitere Methoden ... */
}
```

Aufgabe 2 | **Aufgabe 2: von 10 P**

(a) Welcher Typ muss anstelle von `XXX` im folgenden Listing eingetragen werden? 2 P

```
import java.util.concurrent.CompletableFuture;
public class Klasse {
    public static void main(String[] args) {
        CompletableFuture<XXX> cf = CompletableFuture
            .supplyAsync(() -> 3)
            .thenApplyAsync((n) -> n + 1)
            .thenApplyAsync((n) -> n * 2.0)
            .thenAcceptAsync((n) -> System.out.println(n));
    }
}
```

`XXX` = _____

(b) Wieviele Threads werden in dem obigen Programm (Aufgabe 2a) verwendet (Mehrfachantwort möglich)? 2 P

(Bewertung: alle richtig: 2 Punkte, 1x falsch: 1 Punkt, mehr als 1x falsch: 0 Punkte)

- ☐ mindestens zwei
- ☐ höchstens 5
- ☐ genau 5
- ☐ genau 6
- ☐ genau 2
- ☐ genau 1
- ☐ das hängt davon ab, wie das Programm gestartet wird und wieviele CPUs bzw. Cores die Hardware hat

(c) Wie können drei `CompletableFuture<Float>`-Objekte asynchron addiert werden? Komplettieren Sie den folgenden Programmtext: 6 P

```
import java.util.concurrent.CompletableFuture;
public class Klasse {
    static CompletableFuture<Float> cf1, cf2, cf3;
    public static void main(String[] args) {
        cf1 = CompletableFuture.supplyAsync(() -> new Float(1.0));
        cf2 = CompletableFuture.supplyAsync(() -> new Float(2.0));
        cf3 = CompletableFuture.supplyAsync(() -> new Float(3.0));
```

Aufgabe 3

Aufgabe 3: von 10 P

(a) Nennen Sie fünf Konzepte (zugrundeliegende Prinzipien bzw. Eigenschaften) des Actor-Modells. 5 P

.....

.....

.....

.....

.....

(b) Beschreiben Sie verbal, wie im folgenden Akka-Code-Beispiel Nachrichten vom Typ `TaskMsg` verarbeitet werden. Beschreiben Sie dabei auch, wohin Antworten auf versendete Nachrichten zurückgesendet werden. Beschreiben Sie ggf. Ihre Annahmen über `SubTask`, `TaskMsg` (insb. `getSubTasks()`) und `WorkerActor`. 3 P

```
import akka.actor.*;
import akka.routing.*;
// ...
public class MasterActor extends AbstractActor {
    private static final int WORKER_NUM = 5;
    private final List<Routee> routees;
    private final Router   router;
    public MasterActor() {
        this.routees = new ArrayList<>();
        for (int i = 0; i < MasterActor.WORKER_NUM; i++) {
            final ActorRef r = getContext().actorOf(Props.create(WorkerActor.class));
            this.routees.add(new ActorRefRoutee(r));
        }
        this.router = new Router(new RoundRobinRoutingLogic(), this.routees);
    }
    @Override
    public Receive createReceive() {
        return receiveBuilder()
                .match(TaskMsg.class, this::handleTaskMsg)
                .match(ResultMsg.class, this::handleSubResultMsg)
                .build();
    }
    private void handleTaskMsg(final TaskMsg msg) {
        for (SubTask subtask : msg.getSubTasks()) {
            this.router.route(subtask, getSelf());
        }
    }
    // ...
}
```

(c) Beschreiben Sie wie sich im vorigen Beispiel die Verwendung eines `SmallestMailbox-RoutingLogic`-Routers auf das Verhalten auswirken würde. 1 P

(d) Unter welcher Bedingung sollte besser ein `SmallestMailboxRoutingLogic`-Router statt eines `RoundRobinRoutingLogic`-Router benutzt werden? 1 P

Reserveplatz