hochschule mannheim

PR2 – Programmierung 2 FAKULTÄT FÜR INFORMATIK
Cyber Security Bachelor (CSB), Medizinische Informatik Bachelor (IMB)
Sommersemester 2020 [22.04.2020]

PR2 Aufgabenblatt für Testate Nr. 2

Allgemeine Hinweise

Ausgabe der Übung:	Mi, 22.04.2020, 15:00 Uhr
Abgabe:	Mo, 04.05.2020, 16:00 Uhr
Testat am:	Di, 05.05.2020, 13:40–16:50 (online)

- Die Abnahme der Übungen gilt als **Prüfungsleistung**. Bei einer Verhinderung durch Krankheit ist eine ärztliche Bescheinigung der Arbeitsunfähigkeit vorzulegen.
- Laden Sie Ihre Lösungen bis zur Deadline in Moodle bei der entsprechenden Unteraufgabe hoch. Quellcodes müssen einheitlich und sinnvoll formatiert sein (vorzugsweise mithilfe Ihrer IDE wie Eclipse oder IntelliJ). **Ausarbeitungen in einem anderen Format werden nicht berücksichtigt.**
- Während der Abnahme sind die Ergebnisse am Rechner live zu demonstrieren.
- Bei der Abnahme der Übung ist der Studentenausweis vorzulegen.

Lernziele

- Stream-Klassen für Input/Output in Java anwenden können.
- Nebenläufigkeit in Java umsetzen können
- das Collection-Framework benutzen können
- generische Typen anwenden können

Aufgabe 1
Kopierprogramm

Programmieren Sie ein Programm zum zeichenweisen (nicht in Blöcken) Kopieren: Alle Bytes der Ursprungsdatei sollen in der Reihenfolge ihres Vorkommens in der Ursprungsdatei in eine neue Zieldatei kopiert geschrieben werden. Die Ursprungsdatei darf nicht geändert werden.

Sowohl der Name der Datei, aus der Zeichen gelesen werden sollen („Ursprungsdatei“), als auch der Name der Datei, in die Zeichen geschrieben werden sollen („Zieldatei“), sollen über die Konsole eingegeben werden. Bei Fehlersituationen soll das Programm

eine passende Fehlermeldung auf dem Standard-Fehlerkanal (`stderr`) ausgeben und terminieren. Ist die Ursprungsdatei nicht vorhanden, nicht lesbar oder keine gewöhnliche Datei (sondern z. B. ein Verzeichnis), ist das eine Fehlersituation, die abgefangen werden muss. Falls die Zieldatei bereits existiert oder nicht angelegt werden kann, soll das Programm ebenfalls mit einer Fehlerausgabe auf Standard-Error beendet werden.

Das Programm soll aus einer Klasse bestehen, die im Package `pr2.pu2` liegt und `Copy` heißt.

Aufgabe 2
Nebenläufige Tiere

Programmieren Sie eine Klasse namens `Tier` im Package `pr2.pu2`. Tiere haben eine räumliche Position, die durch eine x- und eine y-Koordinate (vom Typ `int`) repräsentiert wird.

Außerdem muss jedes Tier eine `int`-Instanzvariable haben, die die Lebensenergie repräsentiert. Wenn der Wert dieser Variable `0` oder kleiner ist, hört das `Tier` auf zu leben. Leben bedeutet in diesem Fall, dass die `Tier`-Objekte nebenläufig sind. Ihr Leben besteht aus einer Schleife. In jedem Schleifendurchlauf sinkt die Lebensenergie um `1`. Am Ende jedes Schleifendurchlaufs schläft jedes Tier für 100 ms mit `Thread.sleep(100)`.

Weiterhin braucht die `Tier`-Klasse noch eine Instanzvariable, in der eine Referenz auf den eigenen `Thread` gespeichert wird und einen Zufallszahlengenerator als Klassenvariable.

Alle Instanz-, Klassenvariablen und der Konstruktor sollen die Sichtbarkeit `protected` haben.

Der Konstruktor soll einen `int`-Parameter als Wert für die initiale Lebenskraft haben.

Die Klasse soll eine `main`-Methode haben, in der ein Tier mit der Lebenskraft `33` so erzeugt wird, dass es nebenläufig zum `main`-Thread arbeitet.

Aufgabe 3
Factory-Methode für Tier

Erweitern Sie die Klasse `Tier` um eine Factory-Methode namens `create` mit der Sichtbarkeit `public`, die `static` sein muss. Das Ergebnis dieser Methode ist eine neue `Tier`-Instanz. Das nebenläufige Leben der `Tier`-Instanz soll außerdem bereits in `create` gestartet werden. Das neue Tier soll dabei mit einer zufälligen Lebenskraft zwischen 0 und 99 erzeugt werden.

In einer `static`-Variable vom Typ `int` soll mitgezählt werden, wieviele Instanzen der Klasse bereits erzeugt wurden. Außerdem soll dem Thread mit der Methode `setName` der Name „`Tier-XXX`“ gegeben werden, wobei `XXX` eine fortlaufende Zahl ist: Das erste mit `create` erzeugte Tier soll „`Tier-001`“, das zweite „`Tier-002`“ usw. heißen.

Wenn Sie Ihre Lösung testen, sollten Sie einige Tiere aus getrennten Threads heraus erzeugen und den Namen prüfen.

Aufgabe 4
Subklasse für `Tier`

Schreiben Sie eine neue Klasse namens `Hase` im Package `pr2.pu2`, die von `Tier` erbt. Auch Instanzen von `Hase` sollen nebenläufig sein und `create` soll nun `Hase`- statt `Tier`-Objekte liefern. Die anfängliche Lebenskraft von Hasen soll immer `40` sein.

Aufgabe 5
Tiere in Bewegung

Die `Tier`-Klasse soll um die Möglichkeit zur Bewegung erweitert werden. Dazu muss die Methode `move` programmiert werden. Sie soll die Sichtbarkeit `public` haben.

In ihr soll eine Zufallszahl zwischen 0 und einschließlich 4 gezogen werden. Je nachdem welche Zahl gezogen wird, soll `x` oder `y` um 1 geändert (`+`/`-`) werden oder das Tier soll stehenbleiben.

Zum Lebenszyklus eines `Tier`-Objekts soll die Bewegung mit der `move`-Methode hinzugefügt werden.

Achten Sie bei der Umsetzung darauf, dass die Methode auch von anderen Threads aufgerufen werden kann.

Aufgabe 6
Welche `Collection` als Gehege?

Schreiben Sie den (parametrierten) Typ einer `Collection` auf, in der Sie ein Gehege mit `Tier`-Instanzen repräsentieren würden. Es gibt für diese Teilaufgabe keine genauere Spezifikation, welche Tiere in dem Gehege gespeichert werden sollen.

Begründen Sie in ein bis zwei Sätzen, welche Art von `Collection` Sie dafür verwenden.

Aufgabe 7
Typ-Parameter für nach Tierarten getrennte Gehege

Programmieren Sie die Klasse `ArtenGehege` im Package `pr2.pu2`. Das ist ein Gehege für Tiere. Es dürfen aber nur gleichartige Tiere, also Instanzen desselben Typs bzw. Untertyps von `Tier` darin gespeichert werden. Die Klasse soll als öffentliche Methode `einsperren` anbieten, die ein Tier des entsprechenden Typs als einzigen Parameter erwartet und das übergebene Tier dann in der passenden Datenstruktur (s. Teilaufgabe (7)) speichert.

Schreiben Sie eine `main`-Methode, die einen Hasenstall erzeugt und mit drei Hasen füllt. Der Compiler soll verbieten, ein `Tier`-Objekt, das kein `Hase` ist, in den Hasenstall einzusperren.

Aufgabe 8
`Iterator<Tier>` für den ganzen Zoo

Der ganze Zoo ist als `Map` realisiert. Als Schlüssel dienen die Namen der Gehege, die sich die Direktion des Zoos ausgedacht hat. Die Namen sind vom Typ `String`. Die Werte, die in der `Map` gespeichert sind, sind Instanzen von `ArtenGehege`. Programmieren

Sie die Klasse `Zoo`, die das `Iterable`-Interface implementieren soll. Der dazugehörende `Iterator` soll die `Tier`-Instanzen des Zoos liefern. Die Reihenfolge der Tiere ist dabei egal.

Aufgabe 9
Sortieren nach eigenem Sortierkriterium

Ändern Sie den `Iterator` so ab, dass die `Tier`-Instanzen in einer bestimmten Reihenfolge geliefert werden.

Verwenden Sie zum Sortieren die statische Methode `sort` der Klasse `Collections`. Damit können Sie eine `List` mit einem eigenen `Comparator` sortieren.

Als Sortierkriterium soll der Wert der Lebenskraft verwendet werden (in absteigender Reihenfolge, also zuerst die Tiere mit der größten Lebenskraft). Sie müssen dafür eine eigene `Comparator`-Klasse schreiben, die Tiere miteinander vergleichen kann.

Aufgabe 10
Live-Testat

Wird während des Testats bekannt gegeben.