Vorlesung Software-Projekt

Sommersemester 2014

Prof. Dr. Rainer Koschke

4. Übungsblatt

Dieses Übungsblatt ist spätestens am 13. Juli 2014 23:59 Uhr (MESZ) über MEMS abzugeben.

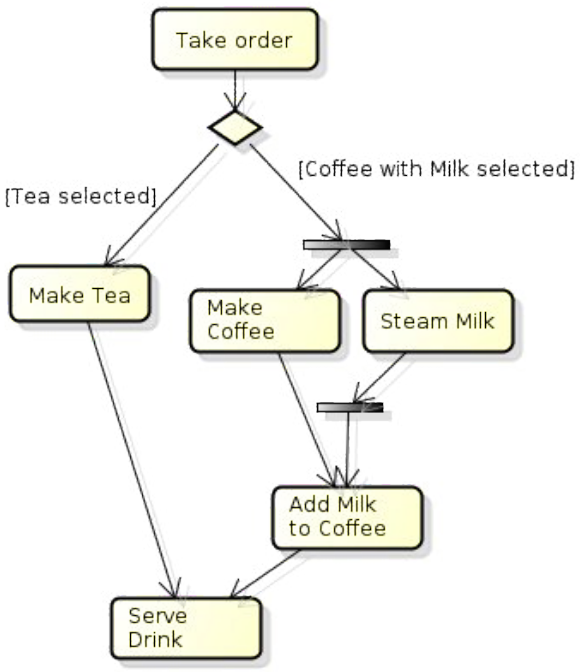
Aufgabe 1 (25 Punkte)

Geben Sie ein Aktivitätsdiagramm für den Kontrollfluss der folgenden Java-Funktion an:

```
int binarySearch(int[] sorted, int key)
{
  int first = 0;
  int last = sorted.length;
  while (first < last)
  {
    int mid = (first + last) / 2;
    if (key < sorted[mid])
      last = mid;
    else if (key > sorted[mid])
      first = mid + 1;
    else
      return mid;
  }
  return -(first + 1);
}
```

Aufgabe 2 (30 Punkte)

Folgendes Aktivitätsdiagramm beschreibt die Tätigkeiten eines Kaffeeautomaten. Die Maschine nimmt einen Getränkewunsch entgegen, bereitet dann den Milchkaffee oder Tee und serviert dann das Getränk in einer Tasse.



a. Das Diagramm hat Mängel. Beschreiben Sie das Diagramm und erläutern Sie die identifizierten Mängel.

b. Beheben Sie die Fehler und geben Sie ein korrigiertes Aktivitätsdiagramm ab.

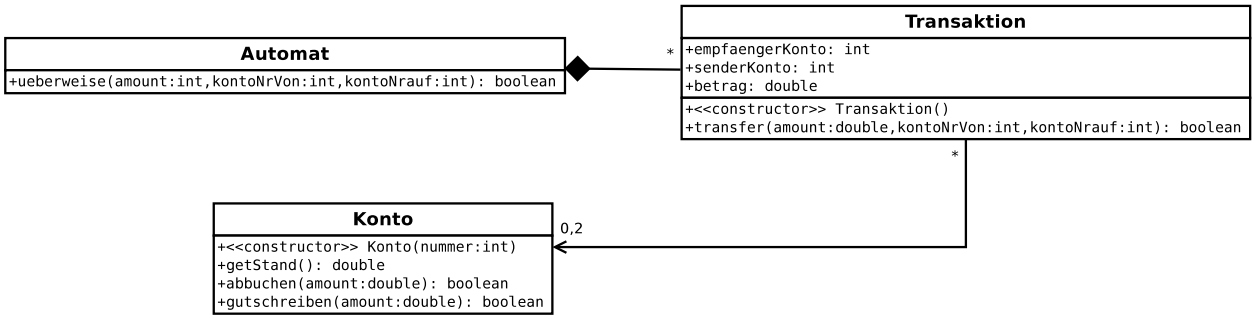
c. Der folgende Quelltext ist unvollständig (bei Stud.IP finden Sie den Quelltext als Java-Datei). Vervollständigen Sie ihn an den mit ... markierten Stellen, so dass das Programm die Aktivitäten aus ihrem korrigierten Aktivitätsdiagramm ausführt. Geben Sie Ihre Lösung als Java-Datei(en) ab, die kompilierbar sind.

```
import java.util.Scanner;

class CoffeeMachine {

  public static void main(String[] args) throws InterruptedException
  {
      ...
  }
}

abstract class Beverage {
    abstract void prepare();
    void mix(Beverage b) {}
    protected void work(int iterations, int ms, String message) {
        for(int i = 1; i <= iterations; i++) {
            System.out.println(message);
            try { Thread.sleep(ms); } catch (Exception e) {}
        }
    }
}

class Tea extends Beverage {
    void prepare() { work(10, 500, "preparing tea"); }
}

class Coffee extends Beverage {
    void prepare() { work(15, 5000, "preparing coffee"); }
}

class Milk extends Beverage {
    void prepare() { work(5, 5000, "preparing milk"); }
}

class Guest {
    public void drink(Beverage beverage) {}
    public boolean wantsTea() {
        Scanner reader = new Scanner(System.in);
        do {
            System.out.println("What is your order: tea or coffee with milk?");
            final String input = reader.nextLine();
            if (input.equals("tea")) {
                return true;
            } else if (input.equals("coffee")) {
                return false;
            }
        } while (true);
    }
}
```

```
class BeverageMaker extends Thread
{
    public Beverage getBeverage() { ... }

    ...
 }
```

Aufgabe 3 (25 Punkte)

Gegeben ist das folgende Klassendiagramm.

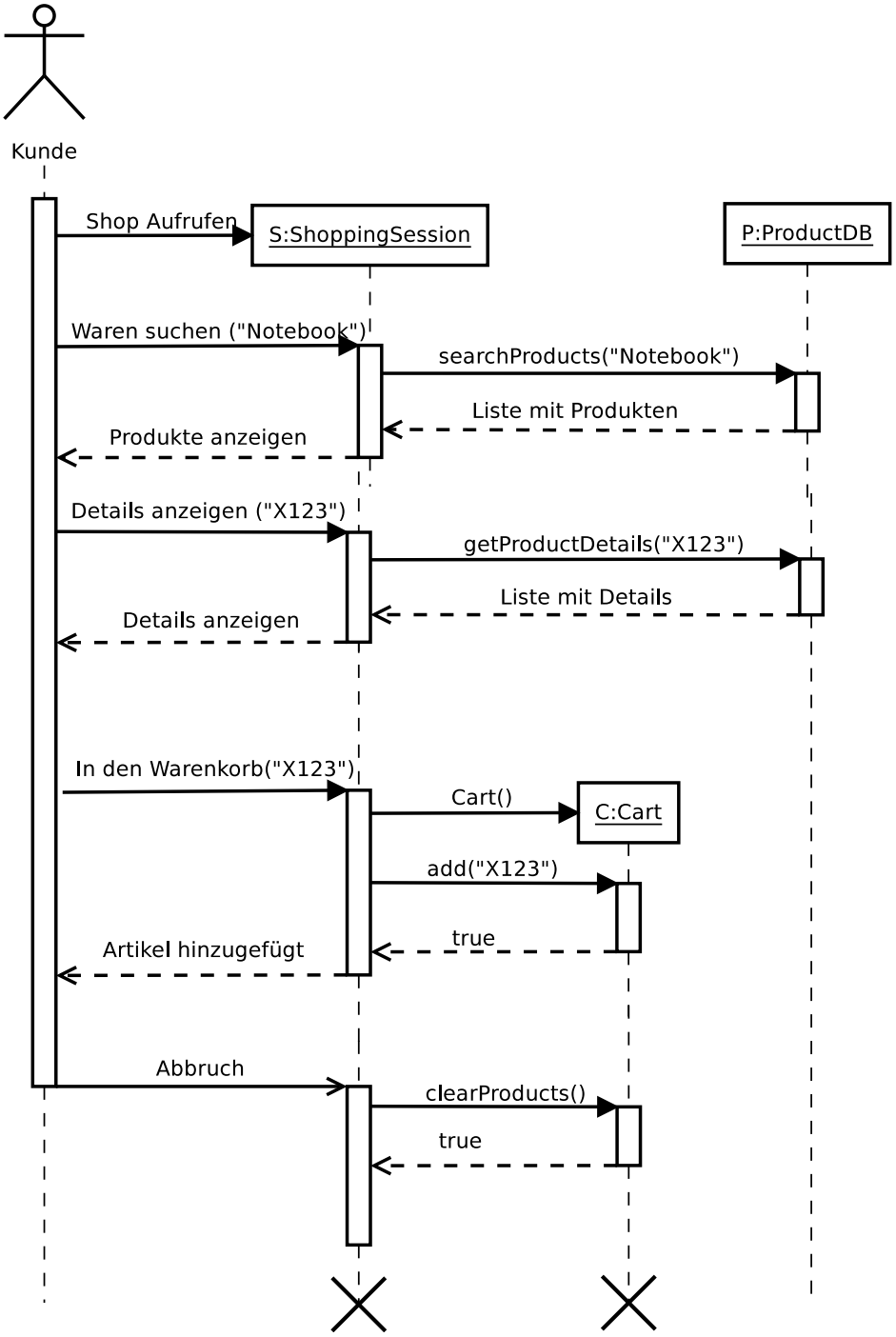


a) Erstellen Sie ein Sequenzdiagramm für den folgenden Ablauf. Ein Kunde überweist am Automaten 100 € vom Konto 123 auf das Konto 234. Der Stand von Konto 123 beträgt 150 €. Stellen Sie in Ihrem Diagramm die Abläufe zwischen allen Klassen und dem Kunden dar. Als Nachrichten verwenden Sie nur die Methoden der Klassen und deren Rückgaben. Der Kunde kann nur Methoden der Klasse *Automat* aufrufen. Stellen Sie sicher, dass das Transaktionsobjekt den Kontostand prüft, bevor ein Konto belastet wird!

b) Stellen Sie in einem zweiten Diagramm das gleiche Szenario dar, allerdings mit dem Unterschied, dass das Konto 123 nur mit 50 € gedeckt ist.

Aufgabe 4 (20 Punkte)

Folgendes Sequenzdiagramm stellt einen Ablauf in einem Online-Shop dar. Die *ShoppingSession* regelt eine Shop-Sitzung und ist der Einfachheit halber auch für die Darstellung des Shops zuständig. Die Klasse *Cart* modelliert einen virtuellen Einkaufswagen, während *ProductDB* eine Zugriffsklasse auf die Produktdatenbank ist.



Beschreiben Sie den abgebildeten Vorgang und zeichnen Sie ein entsprechendes Kommunikationsdiagramm dazu.