Klausur: Einführung in die Softwareentwicklung

Sommersemester 2024 · 02. August 2024

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Angaben zur Person

Vorname	Matrikelnummer

Nachname

Anleitung

Bitte lesen Sie sich die untenstehenden Hinweise **sorgfältig** durch, und folgen Sie der Anleitung!

- **Sollten Sie sich gesundheitlich nicht in der Lage fühlen**, die Prüfung anzutreten, dann **melden Sie sich vor Beginn der Bearbeitung bei der Aufsicht**. Dies ist die letzte Möglichkeit für einen Rücktritt von der Prüfung, und auch nur, wenn ein **ärztliches Attest** im Nachgang vorgelegt werden kann.
- Die Klausur besteht aus **7 Aufgaben** auf insgesamt **13 einzelnen Seiten** (einschließlich dieses Deckblattes). **Prüfen Sie sofort**, ob Sie ein vollständiges Exemplar erhalten haben. Falls etwas nicht stimmt, melden Sie sich sofort, um eine Austauschklausur zu erhalten!
- **Schreiben Sie** Ihre persönlichen Angaben, aber **keine Lösungen** auf das Deckblatt.
- Beantworten Sie die Fragen **direkt auf dem Aufgabenblatt**. Falls Sie mehr Platz benötigen, **nutzen Sie die Rückseite**. Falls dies nicht ausreicht, nutzen Sie zusätzliche Blätter. Melden Sie sich, falls Sie zusätzliches Papier benötigen. Schreiben Sie **auf jedes Blatt, das Lösungen enthält**, als **erstes Ihre Matrikelnummer** (und keine weiteren persönlichen Informationen) **rechts oben** auf die Vorderseite. **Blätter ohne Matrikelnummer werden u.U. nicht gewertet!**
- **Sie können maximal 100 Punkte erreichen.** Sie haben ab 50 Punkten auf jeden Fall bestanden.
- Sie können die Fragen auf **Deutsch** oder **Englisch** beantworten.
- Auf **Wunsch** können (prioritär nicht-Muttersprachler) eine **englische Zusammenfassung** der Klausur von der Aufsicht erhalten. Dies dient als sprachliche Unterstützung – der deutsche Text ist im Zweifelsfall der definitive. Falls Sie eine Zusammenfassung erhalten möchten, melden Sie sich bitte direkt am Anfang der Klausur.
- Ihre abgegebene **Lösung muss klar identifizierbar** sein. Streichen Sie irrelevantes immer durch!
- Sie haben **3 Stunden Zeit** für die Bearbeitung der Klausur.
- Sie dürfen **zwei von Ihnen selbst handbeschriebene Din A4 Blätter** als Hilfsmittel nutzen. Weitere Hilfsmittel (außer Schreibwerkzeug) sind verboten.

Wir wünschen Ihnen viel Erfolg bei der Klausur!

Teil A (Aufgaben 1-5): Programmieraufgaben

Regeln für Teil A:

- Alle Aufgaben können dürfen grundsätzlich in **Python** oder in **Scala** gelöst werden. Sie können dies für jede Aufgabe separat auswählen. **Ausnahme ist 4b**, die Scala verlangt.
- In **Aufgabe 4** können Sie **entweder 4a** (GUI Program.) oder **4b** (Scalaprogram.) wählen. Aufgabe 4b ist für Wiederholer/innen konzipiert, steht aber allen Klausurteilnehmer/innen offen.
- **Sprachregelung:** „Array" bezeichnet Datentypen mit indizierten Feldern, inkl. Python-Typ `list` und Scala's `Array`/`ArrayBuffer`. Strings bezeichnen Zeichenketten (`str`/`String`).
- Hinweise zum **Programmierstil**:
  - Achten Sie darauf, **Einrückungen klar** darzustellen! (insbes. für Python!).
  - Falls Sie den genauen Syntax für etwas vergessen haben, können Sie Pseudo-Code (als solcher markiert!) schreiben, um noch Teilpunkte zu erhalten.
    Beispiel: Falls Sie vergessen haben, dass man in Python mit `isinstance(Objekt, Typ)` Klassenzugehörigkeit testen kann, könnten sie auch „Prüfe_Typkompatibilität(Objekt, Typ) # (Pseudocode: Erklärung was der macht)" schreiben.
  - Wir **empfehlen**, **Kommentare** und (insbesondere für Python) **Typannotationen** zu verwenden, wann immer diese den Code klarer machen. Kommentare sind keine Pflicht, **Typannotationen sind nur dann Pflicht, wenn die Aufgabenstellung oder die Programmiersprache dies verlangt**.
  - Python: Alle Inhalt der Standardmodule, `dataclasses`, `abc`, `typing` und `math` können ohne expliziten import direkt (ohne Modulpräfix) genutzt werden.

Aufgabe 1: Spaß mit Arrays, wie in EIP (10 Punkte)

Schreiben Sie ein Unterprogramme `print_a`, dass Array von Strings als Eingabe bekommt (z.B. in Python: `["a-word", "b-word", "", "42", "abc"]`), und alle Einträge auf der Konsole ausgibt, die mit dem Kleinbuchstaben „`a`" beginnen. Leere Zeichenketten sind als Einträge erlaubt und müssen ignoriert werden.

Aufgabe 2: Entwurf einfacher Summentypen (15 Punkte)

In dieser Aufgabe gilt es, ein sehr einfaches Softwareentwurfsproblem zu lösen und zu implementieren. Es gibt hier verschiedene Lösungen.

Ihre Aufgabe ist es, einen **Datentypen** zu definieren (z.B. mit Hilfe ein oder mehrerer Klassen), der entweder **eine Adresse** oder **ein Datum** enthält. Die Adresse besteht aus der Angabe von **Stadt, Straße und Hausnummer** als drei Strings, ein Datum besteht aus **Tag, Monat und Jahr** als ganze Zahlen. Für beide Varianten des Datentyps muss eine Methode oder ein **Unterprogramm bereitstehen**, mit denen man **Instanzen** dieser Typen **erzeugen** kann, z.B. als Konstruktor (Wertebereiche brauchen an keiner Stelle überprüft zu werden).

Schreiben Sie danach ein **Unterprogramm** `print_array_of_data()` (eine Funktion), die ein **Array von Objekten** übergeben bekommt, in der sowohl **Datums- wie auch Adressobjekte** enthalten sein können (aber keine Instanzen anderer Typen), und diese Objekte nacheinander auf der Konsole ausgibt. Dabei soll das Format wie folgt an den Typ angepasst werden:

- 24.12.2024 (für ein Datumsobjekt, das Weihnachten diesen Jahres kodiert)
- Staudingerweg 9, Mainz (für ein Adressobjekt, das die Adresse der JGU Informatik enthält).

Zur Implementation dieser „print“-Funktionalität können Sie auch die Definition der Datentypen selbst so anpassen, wie es Ihnen nötig erscheint (dies ist nicht zwingend nötig). Erklären Sie Ihre Lösung kurz in Kommentaren, wo dies zum Verständnis sinnvoll ist.

Aufgabe 3: 2D Graphik (15 Punkte)

In dieser und der nächsten Aufgabe nutzen wir die imaginäre Graphikbibliothek „`EisT`“, die `Qt` nachempfunden ist und sowohl für Python wie auch Scala zur Verfügung steht. Sie bietet unter anderem die Klasse `EISPainter`, die es erlaubt einzelne Pixel in einem Bild zu setzen. Alle Bilder sind schwarz / weiß; Farben sind daher Boole'sche Werte (`True` / `False`). Initial sind alle Pixel weiß (`False`). Das Koordinatensystem hat, wie üblich, seinen Ursprung in der linken oberen Ecke und die x-Achse zeigt in Pixelschritten nach rechts, die y-Achse in Pixelschritten nach unten (siehe Abbildung unten).

Die Schnittstelle sieht wie folgt aus:

PYTHON	SCALA

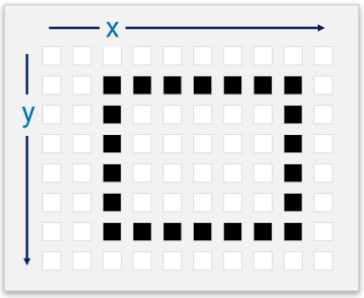
`class EISPainter:`
`  # Ein Pixel setzen (col = Farbe; nur True / False)`
`  def set_pixel(x: int, y: int,`
`  col: bool) -> None`
	`class EISPainter {`
	`  // Ein Pixel setzen (col = Farbe; nur True / False)`
	`  def set_pixel(x: Int, y: Int,`
	`  col: Boolean): Unit`
	`}`

Schreiben Sie ein Unterprogramm

Python: `def draw_rect(p: EISPainter, x: int, y: int, width: int, height: int) -> None`
Scala: `def draw_rect(p: EISPainter, x: Int, y: Int, width: Int, height: Int): Unit`

dass ein schwarzes Rechteck (nicht ausgefüllt) zeichnet, wobei die linke obere Ecke durch `x`, `y` und Breite und Höhe durch `width`, `height` gegeben sind. Die Abbildung unten zeigt ein Beispiel:

Beispiel: x = 2, y = 1, width = 7, height = 6



Aufgabe 4(a): GUI-Programmierung (20 Punkte)

Achtung: Es ist möglich, alternativ zu 4(a) auch Aufgabe 4(b) zu bearbeiten!

Unsere GUI-Bibliothek, bekannt aus Aufgabe 3, unterstützt neben dem `EISPainter` aus Aufgabe 3 auch Widgets mit der Klasse `EISWidget`:

PYTHON

```
class EISWidget(ABC):
    # (1) Maustaste gedrückt
    def mouse_down(x: int, y: int) -> None
    # (2) Maustaste losgelassen
    def mouse_up(x: int, y: int) -> None
    # (3) Maus bewegt (Zustand Maustaste egal)
    def mouse_move(x: int, y: int) -> None
    # (4) Aufruf, wenn Widget sich zeichnen soll
    def paint(p: EISPainter) -> None
```

SCALA

```
abstract class EISWidget {
    // (1) Maustaste gedrückt
    def mouse_down(x: Int, y: Int): Unit;
    // (2) Maustaste losgelassen
    def mouse_up(x: Int, y: Int): Unit;
    // (3) Maus bewegt (Zustand Maustaste egal)
    def mouse_move(x: Int, y: Int): Unit;
    // (4) Aufruf, wenn Widget sich zeichnen soll
    def paint(p: EISPainter): Unit;
}
```

Wie aus den Übungen mit `Qt` bekannt, werden die vier oben beschriebenen Ereignisse automatisch vom `EisT`-Framework aufgerufen, wenn (1) die (linke und einzige) Maustaste über dem Widget gedrückt wurde, (2) die Maustaste wieder losgelassen wurde, (3) der Mauscursor sich über dem Widget bewegt (egal, ob die Taste gedrückt wurde) und (4) das Widget neu gezeichnet werden muss. Um alle sichtbaren Widgets sofort neu zu zeichnen, gibt es eine globale Methode

`def redraw_GUI() -> None` (in Python) bzw. `def redraw_GUI(): Unit` (in Scala)

„`redraw_GUI`“ ruft dabei indirekt die Paint-Methoden aller sichtbaren Widgets mit dem richtigen EISPainter-Objekt auf (ähnlich `QWidet.update()` in Qt, allerdings der Einfachheit halber mit Wirkung auf alle Widgets). Anders als die vier Memberfunktionen ist die globale „`redraw_GUI`“-Funktion bereits fertig implementiert – man soll und kann diese nicht selbst schreiben.

Ihre Aufgabe:

Schreiben Sie ein Widget (Klasse, die von `EISWidget` erbt), das ähnlich zu den Übungen ein einfaches Malprogramm implementiert: Wenn man mit gedrückter Maustaste über das Widget fährt, so werden die Pixel unter dem Mauscursor schwarz eingefärbt. Bewegt man die Maus ohne Druck der Maustaste, passiert nichts. Die Änderungen sollen sofort sichtbar gemacht werden, und auch dann noch erhalten bleiben, wenn das Widget sich neu zeichnet (z.B. nachdem es verdeckt war).

Tipp: Um die Aufgabe zu lösen, ist es notwendig, das Bild, das gemalt wird, im Widget zwischenzuspeichern (**Hinweis:** Eine Klasse analog zu `QImage` gibt es in `EisT` leider nicht.) Die maximale Größe der Zeichenfläche wird 100 x 100 Pixel nie überschreiten.

Mehr Platz für die Lösung von Aufgabe 4(a) oder 4(b)

Aufgabe 4(b): Programmieren mit Scala (20 Punkte)

Sie können **entweder** Aufgabe 4(a) **oder** Aufgabe 4(b) lösen.

Um Aufgabe 4(b) statt 4(a) zu wählen, kreuzen Sie diese Box an:

☐ **Ich wähle Aufgabe 4(b)** statt 4(a) – **I choose Assignment 4(b)** instead of 4(a).

Falls Sie sich umentscheiden bitte die ganze Zeile oben komplett durchstreichen und in Freitext klar sagen, welche Aufgabe sie lösen möchten; falls mehrere gültige Lösungen erkennbar sind, zählt im Zweifelsfall immer die Lösung von 4a).

Diese Aufgabe beschäftigt sich stärker mit Feinheiten der Programmiersprache Scala und wendet sich vor allem an Wiederholer/innen. **Aufgabe 4(b) muss in der Programmiersprache Scala** gelöst werden, Lösungen in anderen Programmiersprachen werden nicht gewertet.

Aufgabe: Scala bietet in der Standardbibliothek Iteratoren an, mit denen man potentiell unendlich lange Sequenzen von Daten modellieren kann, die nur ausgewertet werden, wenn ein Datenobjekt gebraucht wird. Konkret muss man zwei Methoden implementieren, um einen Iterator zu konstruieren:

`Iterator[A]` – Sequenz vom Typ `A` (generischer Typ)
`def hasNext: Boolean` – True genau dann, wenn noch mindestens ein weiteres Element verfügbar ist
`def next(): A` – Fragt das nächste Element ab und bewegt den Iterator ein Element vorwärts

Schreiben Sie nun eine Iteratorklasse, die alle Primzahlen aufzählt, startend mit der Zahl `2`. Der Elementtyp soll dabei „`Int`“ sein.

Aufgabe 5: Serialisierung (20 Punkte)

In dieser Aufgabe sollen Sie eine Funktion schreiben, die Bäume von Objekten in einen String, und wieder zurück, verwandeln kann. Eine solche Funktion könnte man für das Laden und Speichern von Objekten nutzen; wir abstrahieren hier durch die Strings von den Details des Dateizugriffs.

Bei den Objekten, die serialisiert werden sollen, handelt es sich um Primitive eines Zeichenprogramms, ähnlich zu den Übungen. Es bietet nur drei Typen von Formen: zwei Primitive (Kreise und Sterne) sowie hierarchische Gruppen (dies sind einfach Arrays von Shapes):

PYTHON

```
class Shape(ABC): # abstrakte Oberklasse
  pass

@dataclass
class Circle(Shape): # ein Kreis
  x: int
  y: int
  radius: int

@dataclass
class Star(Shape): # ein Stern, immer 7 Ecken
  x: int
  y: int
  radius: int

# Eine Liste/Gruppe von mehreren Shapes
@dataclass
class ShapeList(Shape): #
  children: List['Shape']
```

SCALA

```
// Abstrakte Oberklasse
abstract class Shape;

// ein Kreis
class Circle(val x: Int, val y: Int,
             val radius: Int)
extends Shape;

// ein Stern
class Star(val x: Int, val y: Int,
           val radius: Int)
extends Shape;

// Eine Liste/Gruppe von mehreren Shapes
class ShapeList(var children:
                        Array[Shape])
extends Shape;
```

Zusätzlich gilt: Es gibt grundsätzlich keine zyklischen Referenzen; alle Shape(List)s sind Bäume.

(i) Schreiben Sie eine Funktion `save(s: Shape) -> str` (Pyton) bzw. `save(s: Shape): String` (Scala), die Bäume von Shapes, wie oben definiert, so in einen String umwandelt, dass man daraus den Objektbaum wieder eindeutig rekonstruieren kann.

(ii) Schreiben Sie eine Funktion `load(f: str) -> Shape` (Pyton) bzw. `load(f: String): Shape` (Scala), die Bäume von Shapes, die mit Ihrer Save-Funktion gespeichert wurden, wieder in einen äquivalenten Objektbaum zurückverwandelt.

Falls Sie die Definition der Klassen zur Implementation ändern möchten (dies ist nicht nötig), dann schreiben Sie den obigen Code bitte nochmal ab bzw. komplett in geänderter Form auf.

Auf der nächsten Seite finden Sie noch einige Tipps zur Stringverarbeitung in Python und Scala.

Stringverarbeitung in Python: (Man braucht nicht unbedingt alles für die Lösung.)

- Stringkonstanten mit Hochkomma (`'abc'`) oder Anführungszeichen (`"abc"`).
- Lesezugriff via Eckige-Klammern, Indiziert von Null an: `'42'[1]` ergibt `'2'`.
- Erinnerung: Strings in Python sind unveränderlich.
- Länge via Funktion `len()`: `len('42')` ergibt `2`.
- Zusammensetzen mit „`+`“ oder „`+=`“ Operator: `'42'+'abc'` ergibt `'42abc'`.
- Umwandung von `int` -> `str`: `str(42)` ergibt `'42'`.
- Umwandung von `str` -> `int`: `int('42')` ergibt `42`.
- Aufteilen nach Whitespaces mit Funktion `split()`. „`'42 abc  qrz \n wat'.split()`“, ergibt `['42', 'abc', 'qrz', 'wat']`. Man kann auch andere Trennzeichen an Split als Argument übergeben.

Stringverarbeitung in Scala: (Man braucht nicht unbedingt alles für die Lösung.)

- Stringkonstanten mit Anführungszeichen (`"abc"`). Einzelne Zeichen mit Hochkomma (`'a'`).
- Lesezugriff via Eckige-Klammern, Indiziert von Null an: `"42"[1]` ergibt `"2"`.
- Erinnerung: Strings in Scala sind unveränderlich.
- Länge via Memberfunktion `length()`: `"42".length()` ergibt `2`.
- Zusammensetzen mit „`+`“ oder „`+=`“ Operator: `"42"+"abc"` ergibt `"42abc"`.
- Umwandung von `int` -> `str`: `42.toString`, ergibt `"42"`.
- Umwandung von `str` -> `int`: `"42".toInt`, ergibt `42`.
- Aufteilen nach Leerzeichen mit Memberfunktion `split(" ")`.
  „`"42 abc  qrz wat".split(" ")`“, ergibt `['42', 'abc', 'qrz', 'wat']`. Man kann auch andere Trennzeichen an Split als Argument übergeben.

Mehr Platz für die Lösung von Aufgabe 5

Teil B (Aufgaben 6 & 7): Konzeptionelle Fragen

Regeln für Teil B:

- Es handelt sich um Freitextfragen. Antworten Sie mit einer kurzen Erklärung. Bewertet werden richtige Idee, nicht Stil/Eleganz des Textes. Es ist sinnvoll, sich kurz zu fassen (i.d.R. reichen ca. 1-3 Sätze pro Antwort aus).
- Oft gibt es mehrere richtige Lösungen / Antworten. Alle sinnvollen Antworten zählen.
- Sie dürfen mehr Antworten geben als verlangt, allerdings werden objektiv falsche Aussagen negativ gewertet. Beispiel: „Nennen Sie zwei Vorteile von Python“ – die Antwort „Es ist einfach zu benutzen und ausdrucksstark, und CPython ist sogar schneller als C++“ zählt als eine richtige Lösung, da die ersten beiden Antworten zwar richtig sind, die dritte aber falsch.
- Alle Aufgaben verlangen kurze Erklärungen. Die Frage „Nachteile von Python“ darf man nicht einfach mit „*langsam*“ beantworten; hingegen wäre „*langsam weil interpretiert*“ akzeptabel.

Aufgabe 6: Fragen im Freitext zu Entwurfsmustern (10 Punkte)

(i) In der Vorlesung haben wir zwei Ansätze kennengelernt, um Instanzen von Summentypen Typspezifisch zu verarbeiten: Zum einen den prozedural/funktionalen Ansatz des Patten Matching und zum anderen die Nutzung einer virtuellen Methode aus einer Basisklasse, die in verschiedenen Unterklassen überschrieben wird. Nennen Sie jeweils einen Vorteil für jeden der beiden Ansätze, den dieser gegenüber dem jeweils anderen hat (mit kurzer Begründung).

Vorteil virtuelle Methode/dynamic Dispatch (mit Begründung):

Vorteil Patten Matching (mit Begründung):

(ii) In der Vorlesung haben wir das Muster von „lazy-evaluation“ kennengelernt, bei dem Daten erst bei Bedarf berechnet werden. Überlegen Sie sich ein Beispiel, wo man das sinnvoll einsetzen kann, und erklären Sie den Vorteil. Beschreiben Sie danach auch einen möglichen Nachteil des Musters.

Gutes Beispiel für den „lazy-evaluation“, d.h. Berechnung bei Bedarf (Vorteil diskutieren):

Möglicher Nachteil bei der Verwendung von lazy-evaluation (mit Begründung):

(iii) In der Vorlesung haben wir Client-Server Architekturen kennengelernt, bei denen ein Server via Warteschlangen auf Anfragen anderer Prozesse wartet, und diese dann per Nachricht beantwortet. nennen Sie eine Beispielanwendung, wo sich Server-Prozesse das sinnvoll einsetzten lassen, und erklären Sie den/die Vorteil(e) kurz, in 1-3 Sätzen. Man kann sogar komplexe Softwareprojekte aus mehreren Serverkomponenten aufbauen (Microservicearchitekturen). Das kann eine gute Idee sein - nennen Sie hier aber einen *wesentlichen Nachteil* solcher Ansätze.

Gutes Beispiel für den Einsatz von Serverprozessen (Vorteil diskutieren):

Nachteil, Software aus mehreren Serverprozessen aufzubauen (mit Grund):

Aufgabe 7: Fragen im Freitext zu Programmiersprachen (10 Punkte)

(i) Dynamic Dispatch ist in C++ (und auch in Java/Scala) mittels Offsets in virtuelle Methodentabellen (VMTs) implementiert; im Gegensatz dazu nutzt Python „Dictionary-Lookups“, mit denen direkt in der Klasse nach dem Namen der Methode gesucht wird. Nennen Sie einen wesentlichen Vorteil sowie einen wesentlichen Nachteil der C++-Variante. Erklären Sie Ihre Antwort kurz (ca. ein Satz).

Vorteil VMT-Ansatz (mit kurzer Begründung):

Nachteil VMT-Ansatz (mit kurzer Begründung):

(ii) Programmiersprachen wie C++ oder Modula-2 trennen jedes Modul eines Programms in eine Schnittstellendefinition und eine Implementation auf. Python, Scala und Java verlangen nur eine Datei, in der alles auf einmal steht. Nennen Sie je ein Vor- und ein Nachteil (mit kurzer Begründung) dieser Ansätze.

Vorteil Schnittstellen und Implementation in getrennten Dateien (mit kurzer Begründung):

Nachteil Schnittstellen und Implementation in getrennten Dateien (mit kurzer Begründung):

(iii) Reines Python (z.B. der CPython-Interpreter) nutzt Duck-Typing (auch bekannt als „uniformes Typsystem"), bei dem Variablen auf alle Objekte der Sprache verweisen können, und anhand der Namen verfügbarer Members zur Laufzeit entschieden wird, ob ein Objekt für eine Operation geeignet ist. Was sind Vor- und Nachteile dieses Ansatzes? Was kann man praktisch tun, um einige der Nachteile zu vermeiden?

Nennen Sie einen Vorteil von Duck-Typing (mit kurzer Begründung):

Nennen Sie zwei verschiedene Nachteile von Duck-Typing (mit kurzer Begründung):

Was gibt es an Lösungen, um mindestens einen der Nachteile zu umgehen: