Ausnahmebehandlung

In dieser Aufgabe müssen Sie den zur Verfügung gestellten Code um Exceptions erweitern. Der Code basiert auf der Farm-Stable-Aufgabe von UE07, wurde allerdings etwas angepasst. Im Folgenden werden Ihnen die Klassen gezeigt und Hinweise gegeben, um Sie mit dem Code vertraut zu machen. Die gelb markierten Stellen sind jene, die Sie anschließend bearbeiten müssen (nicht markierte Stellen dürfen nicht verändert werden!). Liefern Methoden derzeit einen booleschen Rückgabewert bedeutet `true`, dass die Operation erfolgreich war, `false`, dass etwas schiefgegangen ist.

Lesen Sie sich zuerst die Beschreibung der Klassen durch und beginnen Sie dann mit den eigentlichen Aufgaben auf Seite 3.

- Klasse `Animal`: Diese Klasse repräsentiert ein Tier des Bauernhofs.

```
class Animal {
    String name;
    double foodPerDay;
    Animal(String name, double foodPerDay) { ... }
    Animal(String name) { ... }
    boolean changeFoodRation(double delta) { ... }
    public String toString () { ... }
}
```

- Klasse `Stable`: Diese Klasse repräsentiert einen Stall mit einem Array an Tieren. Die gesamte Logik zum Verarbeiten dieses Arrays ist bereits fertig implementiert und darf nicht verändert werden.

```
class Stable {
    private final int nr;
    private final Animal[] animals;
    private int nAnimals;
    Stable(int nr, int capacity) { ... }
    private int getIndexOfAnimal(Animal a) { ... }
    public boolean addAnimal(Animal a) { ... }
    public boolean removeAnimal(Animal a) { ... }
    public String toString () { ... }
}
```

- Klasse `Farm`: Diese Klasse repräsentiert die Anwendung inklusive eines interaktiven Konsolenmenüs. Die gesamte Ein- und Ausgabelogik ist bereits fertig implementiert, Sie müssen diese nicht ändern. Die Arrays, welche die `Animal`-Objekte und `Stable`-Objekte speichern, sind mit fixer Größe angegeben – Sie können davon ausgehen, dass nie mehr Objekte gespeichert werden, als Platz ist.

```
public class Farm {
    private final String owner;
    private final String city;
    private final Stable[] stables = new Stable[10];
    private int nStables = 0;
    private final Animal[] animals = new Animal[100];
    private int nAnimals = 0;

    Farm(String owner, String city) { ... }
    public boolean switchStable(Animal a, Stable from, Stable to) { ... }
    public void print() { ... }
    private void printStables() { ... }
    private void printAnimals() { ... }
    private void addAnimal(Animal a) { ... }
    private void addStable(Stable s) { ... }
    private Animal readAnimalByIndex() { ... }
    private Stable readStableByIndex(String dInfo) { ... }
    private Stable readStableByIndex() { ... }
    public void run() { ... }
    private static void printMenu() { ... }
    private static Animal readNewAnimal() { ... }
    private static Stable readNewStable() { ... }
    public static Farm createExampleFarm() throws StableException { ... }
}
```

- Klasse `FarmApplication`: Diese Klasse ist die Startapplikation. In der `main`-Methode wird die Funktion `Farm.createExampleFarm()` aufgerufen, welche eine neue Farm mit ein paar Beispielställen und Beispieltieren erstellt und zurückliefert, um das Testen zu erleichtern.

```
class FarmApplication {
  public static void main(String[] args) throws StableException {
    Out.println("Starting␣test␣application␣with␣example␣farm\n");

    Farm farm = Farm.createExampleFarm();

    Out.println("Test␣farm␣created:");
    farm.print();
    Out.println();

    farm.run();
  }
}
```

1. Ausnahmeklassen

Punkte: 5

Lernziel: In dieser Aufgabe sollen Sie die Implementierung von *Ausnahmeklassen* üben.

Bevor Sie mit dem Umarbeiten des bestehenden Codes beginnen, müssen Sie zuerst eigene Exception-Klassen anhand folgender Hierarchie erstellen, die Sie dann später benötigen:

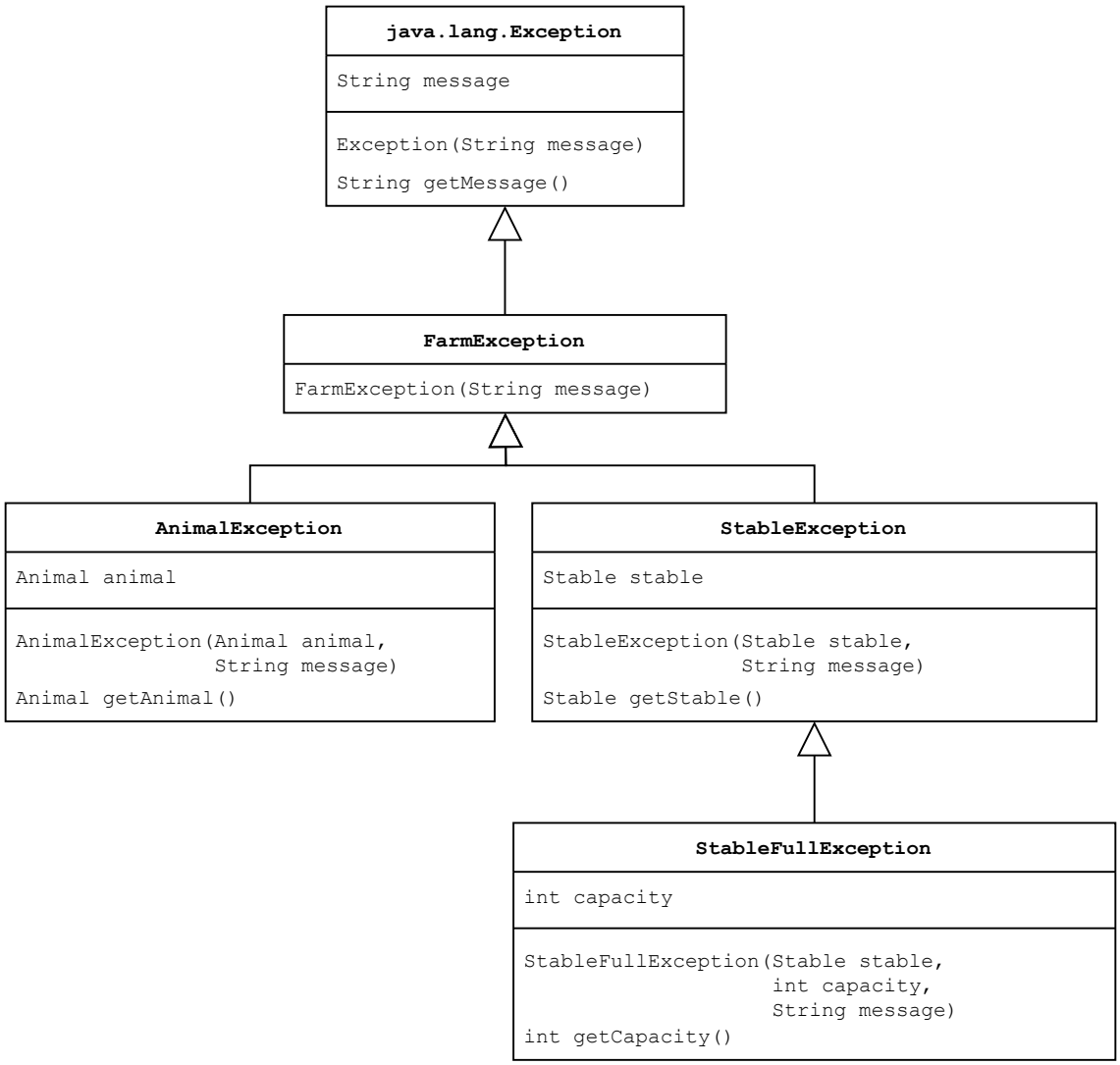

Abbildung 1: Exception-Hierarchie

`FarmException` muss von der (bereits existierenden) Java-Basisklasse `Exception` ableiten und im Konstruktor den Superkonstruktor mit einem String-Parameter (Fehlermeldung) aufrufen. Alle weiteren Implementierungsdetails sind obigem UML-Diagramm zu entnehmen.

2. Werfen und Fangen von Ausnahmen

Punkte: 6

Lernziel: In dieser Aufgabe sollen Sie das *Werfen und Fangen von Ausnahmen* üben.

Beginnen Sie nun, das Programm wie folgt um Exceptions zu erweitern:

Klasse `Animal`:

- Konstruktor `Animal(String name, double foodPerDay)`: Wenn `foodPerDay` ≤ 0 ist, soll eine `java.lang.IllegalArgumentException` mit passender Fehlermeldung geworfen werden.
- Methode `boolean changeFoodRation(double delta)`: Schreiben Sie diese Methode so um, dass sie keinen Rückgabewert mehr hat (`void`) und werfen Sie eine `AnimalException`, wenn die neue Futtermenge ≤ 0 wäre. Übergeben Sie beim Erstellen von `AnimalException` das aktuelle `Animal`-Objekt (`this`) und eine passende Fehlermeldung.

Klasse `Stable`:

- Konstruktor `Stable(int nr, int capacity)`: Wenn `capacity` ≤ 0 ist, soll eine `java.lang.IllegalArgumentI` mit passender Fehlermeldung geworfen werden.
- Methode `boolean addAnimal(Animal a)`: Schreiben Sie diese Methode so um, dass sie keinen Rückgabewert mehr hat (`void`), werfen Sie eine `StableFullException`, wenn im `animals`-Array kein Platz mehr ist (`nAnimals >= animals.length`) und werfen Sie eine `StableException`, wenn das übergebene `Animal`-Objekt `a` bereits im Stable enthalten ist (`getIndexOfAnimal(a)` liefert `!= -1`). Übergeben Sie beim Erstellen der Exceptions jeweils das aktuelle `Stable`-Objekt (`this`) und eine passende Fehlermeldung. Beim Erzeugen einer `StableFullException` übergeben Sie zusätzlich als `capacity` die Länge des `animal`-Arrays.
- Methode `boolean removeAnimal(Animal a)`: Schreiben Sie diese Methode so um, dass sie keinen Rückgabewert mehr hat (`void`) und werfen Sie eine `StableException`, wenn das übergebene `Animal`-Objekt `a` nicht gefunden wurde (`getIndexOfAnimal(a)` liefert `-1`). Übergeben Sie beim Erstellen der Exception das aktuelle `Animal`-Objekt (`this`) und eine passende Fehlermeldung.

Klasse `Farm`:

- Methode `boolean switchStable(Animal a, Stable from, Stable to)`: Schreiben Sie diese Methode so um, dass sie keinen Rückgabewert mehr hat (`void`) und werfen Sie eine `FarmException`, wenn die beiden übergebenen `Stable`-Objekte `from` und `to` auf dasselbe Objekt verweisen (`from == to`). Hinweis: Die Methoden `removeAnimal` und `addAnimal` haben nach obiger Änderung keinen Rückgabewert mehr, d.h., Sie können diese einfach hintereinander als Prozeduren (`void`-Methoden) aufrufen. Mögliche Ausnahmen, die in diesen Methoden geworfen werden, sollen von `switchStable` nach außen weitergeleitet werden.
- Methode `void run()`: Packen Sie das `switch`-Statement (siehe markierter Code oben) in ein `try-catch` und fangen Sie folgende Exceptions ab:
  - `IllegalArgumentException`: Geben Sie auf der Konsole einen Text aus, der die Fehlermeldung der gefangenen Exception (`ex.getMessage()`) enthält.
  - `FarmException`: Geben Sie auf der Konsole einen Text aus, der die Fehlermeldung der gefangenen Exception (`ex.getMessage()`) enthält.
  - `AnimalException`: Geben Sie auf der Konsole einen Text aus, der das `Animal`-Objekt sowie die Fehlermeldung der gefangenen Exception (`ex.getMessage()`) enthält.

- `StableException`: Geben Sie auf der Konsole einen Text aus, der das `Stable`-Objekt sowie die Fehlermeldung der gefangenen Exception (`ex.getMessage()`) enthält.
- `StableFullException`: Geben Sie auf der Konsole einen Text aus, der das `Stable`-Objekt sowie die Fehlermeldung der gefangenen Exception (`ex.getMessage()`) enthält.

Achten Sie auf die korrekte Reihenfolge der `catch`-Klauseln.

Zusätzliche Anweisungen

- Übergeben Sie bei der Erstellung *aller* Exceptions sinnvolle Fehlermeldungen.
- Achten Sie darauf, passende `throws`-Klauseln zu definieren.

3. Testen des umgeschriebenen Programms

Punkte: 3

Testen Sie Ihr neues Programm mittels diverser Operationen im zur Verfügung gestellten Konsolenmenü (nutzen Sie hierfür auch die bereits hinzugefügten `Animal`-Objekte und `Stable`-Objekte). Testen Sie dabei *jeden möglichen Fehlerfall* mindestens einmal, d.h., jede mögliche Exception muss zumindest einmal auftreten. Eine Beispielausgabe (welche mit abzugeben ist) könnte wie folgt aussehen:

```
Starting test application with example farm

Test farm created:
Stables:
[0] Stable #1 (capacity 2/2)
- Animal Blinky (Food per day: 1,00)
- Animal Susi (Food per day: 50,00)
[1] Stable #2 (capacity 1/5)
- Animal Berta (Food per day: 52,50)
Animals:
[0] Animal Blinky (Food per day: 1,00)
[1] Animal Susi (Food per day: 50,00)
[2] Animal Berta (Food per day: 52,50)
[3] Animal Ducky (Food per day: 2,50)

============== Menu ==============
Print farm ..................... p
Add new animal to farm ....... a
Add new stable to farm ....... s
Assign animal to stable ...... +
Remove animal from stable .... -
Switch animal's stable........ c
Change animal's daily food ... f
Exit application ............. x
Select operation: a
Enter animal name: Fred
Enter daily food: -3
Some argument was wrong.
Error message: food per day must be > 0

============== Menu ==============
Print farm ..................... p
Add new animal to farm ....... a
Add new stable to farm ....... s
Assign animal to stable ...... +
Remove animal from stable .... -
Switch animal's stable........ c
Change animal's daily food ... f
Exit application ............. x
Select operation: s
Enter stable number: 3
Enter stable capacity: 0
Some argument was wrong.
Error message: capacity must be > 0

============== Menu ==============
Print farm ..................... p
Add new animal to farm ....... a
Add new stable to farm ....... s
Assign animal to stable ...... +
Remove animal from stable .... -
Switch animal's stable........ c
```

```
Change animal's daily food ... f
Exit application ..... x
Select operation: p
Stables:
[0] Stable #1 (capacity 2/2)
- Animal Blinky (Food per day: 1,00)
- Animal Susi (Food per day: 50,00)
[1] Stable #2 (capacity 1/5)
- Animal Berta (Food per day: 52,50)
Animals:
[0] Animal Blinky (Food per day: 1,00)
[1] Animal Susi (Food per day: 50,00)
[2] Animal Berta (Food per day: 52,50)
[3] Animal Ducky (Food per day: 2,50)

=============== Menu ===============
Print farm ..... p
Add new animal to farm ..... a
Add new stable to farm ..... s
Assign animal to stable ..... +
Remove animal from stable .... -
Switch animal's stable..... c
Change animal's daily food ... f
Exit application ..... x
Select operation: +
Animals:
[0] Animal Blinky (Food per day: 1,00)
[1] Animal Susi (Food per day: 50,00)
[2] Animal Berta (Food per day: 52,50)
[3] Animal Ducky (Food per day: 2,50)
Select animal (index): 2
Stables:
[0] Stable #1 (capacity 2/2)
- Animal Blinky (Food per day: 1,00)
- Animal Susi (Food per day: 50,00)
[1] Stable #2 (capacity 1/5)
- Animal Berta (Food per day: 52,50)
Select stable (index): 1
Something went wrong for stable
Stable #2 (capacity 1/5)
- Animal Berta (Food per day: 52,50).
Error message: stable already contains Animal Berta (Food per day: 52,50)

=============== Menu ===============
Print farm ..... p
Add new animal to farm ..... a
Add new stable to farm ..... s
Assign animal to stable ..... +
Remove animal from stable .... -
Switch animal's stable..... c
Change animal's daily food ... f
Exit application ..... x
Select operation: +
Animals:
[0] Animal Blinky (Food per day: 1,00)
[1] Animal Susi (Food per day: 50,00)
[2] Animal Berta (Food per day: 52,50)
[3] Animal Ducky (Food per day: 2,50)
Select animal (index): 3
Stables:
[0] Stable #1 (capacity 2/2)
- Animal Blinky (Food per day: 1,00)
- Animal Susi (Food per day: 50,00)
```

```
[1] Stable #2 (capacity 1/5)
- Animal Berta (Food per day: 52,50)
Select stable (index): 0
Something went wrong for stable
Stable #1 (capacity 2/2)
- Animal Blinky (Food per day: 1,00)
- Animal Susi (Food per day: 50,00)
It is already full.
Error message: stable capacity of 2 already reached

============== Menu ==============
Print farm ................... p
Add new animal to farm ....... a
Add new stable to farm ....... s
Assign animal to stable ...... +
Remove animal from stable .... -
Switch animal's stable........ c
Change animal's daily food ... f
Exit application ............. x
Select operation: -
Animals:
[0] Animal Blinky (Food per day: 1,00)
[1] Animal Susi (Food per day: 50,00)
[2] Animal Berta (Food per day: 52,50)
[3] Animal Ducky (Food per day: 2,50)
Select animal (index): 0
Stables:
[0] Stable #1 (capacity 2/2)
- Animal Blinky (Food per day: 1,00)
- Animal Susi (Food per day: 50,00)
[1] Stable #2 (capacity 1/5)
- Animal Berta (Food per day: 52,50)
Select stable (index): 1
Something went wrong for stable
Stable #2 (capacity 1/5)
- Animal Berta (Food per day: 52,50).
Error message: stable does not contain Animal Blinky (Food per day: 1,00)

============== Menu ==============
Print farm ................... p
Add new animal to farm ....... a
Add new stable to farm ....... s
Assign animal to stable ...... +
Remove animal from stable .... -
Switch animal's stable........ c
Change animal's daily food ... f
Exit application ............. x
Select operation: c
Animals:
[0] Animal Blinky (Food per day: 1,00)
[1] Animal Susi (Food per day: 50,00)
[2] Animal Berta (Food per day: 52,50)
[3] Animal Ducky (Food per day: 2,50)
Select animal (index): 0
Stables:
[0] Stable #1 (capacity 2/2)
- Animal Blinky (Food per day: 1,00)
- Animal Susi (Food per day: 50,00)
[1] Stable #2 (capacity 1/5)
- Animal Berta (Food per day: 52,50)
Select from stable (index): 1
Select to stable (index): 0
Something went wrong for stable
```

```
Stable #2 (capacity 1/5)
- Animal Berta (Food per day: 52,50).
Error message: stable does not contain Animal Blinky (Food per day: 1,00)

============== Menu ==============
Print farm ..................... p
Add new animal to farm ....... a
Add new stable to farm ....... s
Assign animal to stable ...... +
Remove animal from stable .... -
Switch animal's stable........ c
Change animal's daily food ... f
Exit application ............. x
Select operation: c
Animals:
[0] Animal Blinky (Food per day: 1,00)
[1] Animal Susi (Food per day: 50,00)
[2] Animal Berta (Food per day: 52,50)
[3] Animal Ducky (Food per day: 2,50)
Select animal (index): 2
Stables:
[0] Stable #1 (capacity 2/2)
- Animal Blinky (Food per day: 1,00)
- Animal Susi (Food per day: 50,00)
[1] Stable #2 (capacity 1/5)
- Animal Berta (Food per day: 52,50)
Select from stable (index): 1
Select to stable (index): 0
Something went wrong for stable
Stable #1 (capacity 2/2)
- Animal Blinky (Food per day: 1,00)
- Animal Susi (Food per day: 50,00)
It is already full.
Error message: stable capacity of 2 already reached

============== Menu ==============
Print farm ..................... p
Add new animal to farm ....... a
Add new stable to farm ....... s
Assign animal to stable ...... +
Remove animal from stable .... -
Switch animal's stable........ c
Change animal's daily food ... f
Exit application ............. x
Select operation: c
Animals:
[0] Animal Blinky (Food per day: 1,00)
[1] Animal Susi (Food per day: 50,00)
[2] Animal Berta (Food per day: 52,50)
[3] Animal Ducky (Food per day: 2,50)
Select animal (index): 0
Stables:
[0] Stable #1 (capacity 2/2)
- Animal Blinky (Food per day: 1,00)
- Animal Susi (Food per day: 50,00)
[1] Stable #2 (capacity 0/5)
Select from stable (index): 0
Select to stable (index): 0
Something went wrong on farm.
Error message: from and to must refer to different stables

============== Menu ==============
Print farm ..................... p
```

```
Add new animal to farm ....... a
Add new stable to farm ....... s
Assign animal to stable ...... +
Remove animal from stable .... -
Switch animal's stable........ c
Change animal's daily food ... f
Exit application ............. x
Select operation: f
Animals:
[0] Animal Blinky (Food per day: 1,00)
[1] Animal Susi (Food per day: 50,00)
[2] Animal Berta (Food per day: 52,50)
[3] Animal Ducky (Food per day: 2,50)
Select animal (index): 0
Change daily food by: -1.5
Something went wrong for animal
Animal Blinky (Food per day: 1,00).
Error message: new food per day must be > 0

=============== Menu ===============
Print farm ................... p
Add new animal to farm ....... a
Add new stable to farm ....... s
Assign animal to stable ...... +
Remove animal from stable .... -
Switch animal's stable........ c
Change animal's daily food ... f
Exit application ............. x
Select operation: x
```

Zusätzliche Anweisungen

- Beachten Sie beim Anlegen neuer `Animal`-Objekte, dass deren Namen nur aus einem einzigen Wort (keine Leerzeichen) bestehen dürfen.
- Der Zugriff auf bestehende `Animal`-Objekte und `Stable`-Objekte erfolgt basierend auf einem Index. Sie können davon ausgehen, dass der Benutzer/die Benutzerin (in diesem Falle Sie) nur gültige Werte eingibt.

4. Fragen zu Ausnahmen

Lernziel: In diesen Aufgaben sollen Sie Ihr *Verständnis von Ausnahmen* vertiefen.

Beantworten Sie die folgenden Verständnisfragen (die gezeigten Beispiele dienen rein dem Verständnis und sollten in der gegebenen Form nicht in "echten" Programmen verwendet werden). Die Klassenhierarchie der Exceptions finden Sie in Abbildung 1. Punkte (`...`) stehen für irrelevanten Code, den Sie als gültig ansehen können.

4.1.

Punkte: 1.5

Gegeben ist folgender Codeausschnitt:

```
int func() {
    int x = 1;
    return calc(x);
}

int calc(int x) throws IllegalArgumentException {
    ...
    if(x >= stables.length) {
        throw new IllegalArgumentException();
    }
    ...
    return 17;
}
```

Ist die Methode `int func()` gültig (kompiliert der Code)? Begründen Sie Ihre Antwort.

4.2.

Punkte: 1.5

Gegeben ist folgender Codeausschnitt:

```
int func() {
    int x = 1;
    return calc(x);
}

int calc(int x) throws FarmException {
    ...
    if(x >= stables.length) {
        throw new FarmException(...);
    }
    ...
    return 17;
}
```

Ist die Methode `int func()` gültig (kompiliert der Code)? Begründen Sie Ihre Antwort.

4.3.

Punkte: 1.5

Gegeben ist folgender Codeausschnitt:

```
int func() {
    int x = 1;
    try {
        return calc(x);
    } catch (StableException e) {
        // handle exception
    }
    return x;
}

int calc(int x) throws FarmException {
    ...
    if(x >= stables.length) {
        throw new FarmException(...);
    }
    ...
}
```

Ist die Methode `int func()` gültig (kompiliert der Code)? Begründen Sie Ihre Antwort.

4.4.

Punkte: 1.5

Gegeben ist folgender Codeausschnitt:

```
int func() {
    int x = 1;
    try {
        return calc(x);
    } catch (StableException e) {
        // handle exception
    }
    return x;
}

int calc(int x) throws StableFullException {
    ...
    if(x >= stables.length) {
        throw new StableFullException(...);
    }
    ...
    return 17;
}
```

Ist die Methode `int func()` gültig (kompiliert der Code)? Begründen Sie Ihre Antwort.

4.5.

Punkte: 4

Gegeben ist folgender Codeausschnitt:

```
void fun1(int x) throws Exception {
    try {
        fun2(x);
        Out.print("a1");
    } catch (StableFullException e) {
        Out.print("b1");
        throw new FarmException(...);
    } catch (StableException e) {
        Out.print("c1");
    }
    Out.print("d1");
}

void fun2(int x) throws Exception {
    try {
        fun3(x);
        Out.print("a2");
    } catch (IllegalArgumentException e) {
        Out.print("b2");
        if (x < -10) {
            throw new AnimalException(...);
        }
        Out.print("c2");
    } finally {
        Out.print("d2");
    }
    Out.print("e2");
}

void fun3(int x) throws Exception {
    if (x < 0) {
        throw new IllegalArgumentException();
    }
    if (x > 10) {
        throw new StableFullException(...);
    }
    Out.print("a3");
}
```

Versuchen Sie, für den obigen Code das Ergebnis vorauszusagen, ohne den Code auszuführen, wenn folgende Methodenaufrufe passieren:

- `fun1(1)`
- `fun1(-1)`
- `fun1(-15)`
- `fun1(15)`

Geben Sie hierfür sowohl die Reihenfolge der Methodenaufrufe (samt Parameterwerten), die produzierte Ausgabe, sowie mögliche ungefangene Ausnahmen an, die zum Rufer von `fun1` weitergeleitet werden. Beispiel (nicht korrekt, nur für Demonstrationszwecke):

- Methodenreihenfolge: `fun1(123)` → `fun2(456)`
- Ausgabe: `x1y1y2z1` mit anschließender `IllegalArgumentException`

Zusätzliche Anweisungen

- Verwenden Sie zur Implementierung der Ein- und Ausgabe für alle Programme nur die Funktionen der beiden bereitgestellten Klassen `In` und `Out`, wie in der Übung gezeigt. Die entsprechenden Java-Dateien samt HTML-Dokumentation sind in Moodle unter *InOut.zip* zu finden.
- Implementieren Sie formatierte Ausgaben, indem Sie `String.format` in Kombination mit `Out.print` und `Out.println` verwenden, wie in der Übung gezeigt. Verwenden Sie als Referenz den Foliensatz `StringFormat.pdf` in Moodle.
- Vermeiden Sie Codeduplikation.
- Verwenden Sie für Variablen passende Datentypen Ihrer Wahl.
- Alle Namen (Klassennamen, Variablennamen, Methodennamen, etc.) sind in Englisch zu wählen.
- Formatierung, Namenswahl, etc. fließen in die Bewertung mit ein.

Abzugeben

Geben Sie *eine .zip Datei* mit dem Namen kxxxxxxxx_UE09.zip ab (Beispiel: k01234567_UE01.zip). Darin muss enthalten sein:

- Der *Source Code* aller Programmieraufgaben der aktuellen Übung (alle `.java` Dateien, keine `.class` Dateien).
- Ein Testprotokoll in Form eines gesamten Testlaufes.
- Ihre Antworten zu den Verständnisfragen.