RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Blatt 7 Gruppe A

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 7 (10 Punkte)

Implementieren Sie eine Methode mit der folgenden Signatur:

public boolean *isInjective*(*java.util.Map*$\langle A, B\rangle$ *map*)

Diese Methode soll genau dann *true* zurückliefern, wenn *map* injektiv ist, das heißt wenn für alle Objekte *a1* und *a2*, für die Werte in *map* hinterlegt sind, gilt:

$a1.equals(a2) \vee !map.get(a1).equals(map.get(a2))$

Die Map *map* soll von der Methode *nicht* verändert werden.

Sie können bei der Lösung dieser Aufgabe davon ausgehen, dass alle Objekte von Typ A oder B sinnvolle Implementierungen der Methoden *hashCode* und *equals* zur Verfügung stellen. Außerdem können Sie davon ausgehen, dass *map* den Wert *null* weder als key noch als value enthält.

RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Blatt 7 Gruppe B

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 7 (10 Punkte)

Implementieren Sie eine Methode mit folgender Signatur:

public *java.util.Set*⟨*A*⟩ *symmetricDifference*(*java.util.Set*⟨*A*⟩ *set1*, *java.util.Set*⟨*A*⟩ *set2*)

Diese Methode soll die symmetrische Differenz der Mengen *set1* und *set2* zurückliefern, das heißt ein Objekt *a* soll genau dann Element der zurückgegebenen Menge sein, wenn es *entweder* Element der Menge *set1*, *oder* Element der Menge *set2* ist. Das heißt insbesondere, dass Objekte, die sowohl Element der Menge *set1* als auch der Menge *set2* sind, nicht Element der zurückgegebenen Menge sein dürfen.

Die Argumente der Methode sollen dabei *nicht* verändert werden.

Sie können bei der Lösung dieser Aufgabe davon ausgehen, dass alle Objekte von Typ *A* sinnvolle Implementierungen der Methoden *hashCode* und *equals* zur Verfügung stellen. Außerdem können Sie davon ausgehen, dass *null* weder Element von *set1* noch von *set2* ist.

RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Blatt 7 Gruppe C

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 7 (10 Punkte)

Implementieren Sie eine Methode mit folgender Signatur:

public *java.util.Set*⟨*A*⟩ *intersect*(*java.util.Set*⟨*A*⟩ *set1*, *java.util.Set*⟨*A*⟩ *set2*)

Diese Methode soll die Schnittmenge der Mengen *set1* und *set2* zurückliefern.

Die Argumente der Methode sollen dabei *nicht* verändert werden.

Sie können bei der Lösung dieser Aufgabe davon ausgehen, dass alle Objekte von Typ A sinnvolle Implementierungen der Methoden *hashCode* und *equals* zur Verfügung stellen. Außerdem können Sie davon ausgehen, dass *null* weder Element von *set1* noch von *set2* ist.

RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Blatt 7 Gruppe D

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 7 (10 Punkte)

Implementieren Sie eine Methode mit der folgenden Signatur:

public boolean $isSurjective(java.util.Map\langle A, B\rangle\ map, java.util.Set\langle B\rangle\ codomain)$

Diese Methode soll genau dann *true* zurückliefern, wenn *map* surjektiv bezüglich des als Argument übergebenen Wertebereichs *codomain* ist, das heißt wenn für alle Elemente $b \in codomain$ ein Objekt a existiert, so dass $b.equals(map.get(a))$ gilt.

Die Argumente der Methode sollen dabei *nicht* verändert werden.

Sie können bei der Lösung dieser Aufgabe davon ausgehen, dass alle Objekte von Typ A oder B sinnvolle Implementierungen der Methoden *hashCode* und *equals* zur Verfügung stellen. Außerdem können Sie davon ausgehen, dass *map* den Wert *null* weder als key noch als value enthält und dass *null* kein Element von *codomain* ist.

RWTH Aachen
Lehrgebiet Theoretische Informatik
Frohn–Reidl–Rossmanith–Sánchez–Ströder

WS 2013/14
Blatt 7 Gruppe E

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 7 (10 Punkte)

Implementieren Sie eine Methode mit folgender Signatur:

public boolean $isSubMap(java.util.Map\langle A, B\rangle\ subMap, java.util.Map\langle A, B\rangle\ superMap)$

Diese Methode soll genau dann *true* zurückliefern, wenn für jedes Paar (k, v), für das $subMap.get(k).equals(v)$ gilt, auch $superMap.get(k).equals(v)$ gilt.

Die Argumente der Methode sollen dabei *nicht* verändert werden.

Sie können bei der Lösung dieser Aufgabe davon ausgehen, dass alle Objekte von Typ A oder B sinnvolle Implementierungen der Methoden *hashCode* und *equals* zur Verfügung stellen. Außerdem können Sie davon ausgehen, dass sowohl *subMap* als auch *superMap* den Wert *null* weder als key noch als value enthalten.

RWTH Aachen **WS 2013/14**
Lehrgebiet Theoretische Informatik Blatt 7 Gruppe A
Frohn–Reidl–Rossmanith–Sánchez–Ströder

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 7 (10 Punkte)

Implementieren Sie eine Methode mit der folgenden Signatur:

public boolean *isInjective*(*java.util.Map*⟨*A*, *B*⟩ *map*)

Diese Methode soll genau dann *true* zurückliefern, wenn *map* injektiv ist, das heißt wenn für alle Objekte *a1* und *a2*, für die Werte in *map* hinterlegt sind, gilt:

$a1.equals(a2) \vee !map.get(a1).equals(map.get(a2))$

Die Map *map* soll von der Methode *nicht* verändert werden.

Sie können bei der Lösung dieser Aufgabe davon ausgehen, dass alle Objekte von Typ A oder B sinnvolle Implementierungen der Methoden *hashCode* und *equals* zur Verfügung stellen. Außerdem können Sie davon ausgehen, dass *map* den Wert *null* weder als key noch als value enthält.

Lösungsvorschlag:

```
public boolean isInjective(Map⟨A, B⟩ map) {
  Set⟨B⟩ seen = new HashSet⟨B⟩();
  for (B value : map.values()) {
    if (seen.contains(value)) {
      return false;
    }
    seen.add(value);
  }
  return true;
}
public boolean isInjectiveAlt(Map⟨A, B⟩ map) {
  return new HashSet⟨B⟩(map.values()).size() == map.size();
}
```

RWTH Aachen **WS 2013/14**
Lehrgebiet Theoretische Informatik Blatt 7 Gruppe B
Frohn–Reidl–Rossmanith–Sánchez–Ströder

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 7 (10 Punkte)

Implementieren Sie eine Methode mit der folgenden Signatur:

public boolean *isSurjective*(*java.util.Map*⟨*A*, *B*⟩ *map*, *java.util.Set*⟨*B*⟩ *codomain*)

Diese Methode soll genau dann *true* zurückliefern, wenn *map* surjektiv bezüglich des als Argument übergebenen Wertebereichs *codomain* ist, das heißt wenn für alle Elemente $b \in codomain$ ein Objekt a existiert, so dass $b.equals(map.get(a))$ gilt.

Die Argumente der Methode sollen dabei *nicht* verändert werden.

Sie können bei der Lösung dieser Aufgabe davon ausgehen, dass alle Objekte von Typ A oder B sinnvolle Implementierungen der Methoden *hashCode* und *equals* zur Verfügung stellen. Außerdem können Sie davon ausgehen, dass *map* den Wert *null* weder als key noch als value enthält und dass *null* kein Element von *codomain* ist.

Lösungsvorschlag:

```
public boolean isSurjektiv(Map⟨A, B⟩ map, Set⟨B⟩ codomain) {
  for (B b : codomain) {
    if (!map.values().contains(b)) {
      return false;
    }
  }
  return true;
}
public boolean isSurjektivAlt(Map⟨A, B⟩ map, Set⟨B⟩ codomain) {
  return map.values().containsAll(codomain);
}
```

RWTH Aachen **WS 2013/14**
Lehrgebiet Theoretische Informatik Blatt 7 Gruppe C
Frohn–Reidl–Rossmanith–Sánchez–Ströder

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 7 (10 Punkte)

Implementieren Sie eine Methode mit folgender Signatur:

public *java.util.Set⟨A⟩ intersect(java.util.Set⟨A⟩ set1, java.util.Set⟨A⟩ set2)*

Diese Methode soll die Schnittmenge der Mengen *set1* und *set2* zurückliefern.

Die Argumente der Methode sollen dabei *nicht* verändert werden.

Sie können bei der Lösung dieser Aufgabe davon ausgehen, dass alle Objekte von Typ A sinnvolle Implementierungen der Methoden *hashCode* und *equals* zur Verfügung stellen. Außerdem können Sie davon ausgehen, dass *null* weder Element von *set1* noch von *set2* ist.

Lösungsvorschlag:

```
public Set⟨A⟩ intersection(Set⟨A⟩ set1, Set⟨A⟩ set2) {
  Set⟨A⟩ res = new HashSet⟨A⟩();
  for (A element : set1) {
    if (set2.contains(element)) {
      res.add(element);
    }
  }
  return res;
}
public Set⟨A⟩ intersectionAlt(Set⟨A⟩ set1, Set⟨A⟩ set2) {
  Set⟨A⟩ res = new HashSet⟨A⟩(set1);
  res.retainAll(set2);
  return res;
}
```

RWTH Aachen **WS 2013/14**
Lehrgebiet Theoretische Informatik Blatt 7 Gruppe D
Frohn–Reidl–Rossmanith–Sánchez–Ströder

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 7 (10 Punkte)

Implementieren Sie eine Methode mit folgender Signatur:

public *java.util.Set⟨A⟩ symmetricDifference(java.util.Set⟨A⟩ set1, java.util.Set⟨A⟩ set2)*

Diese Methode soll die symmetrische Differenz der Mengen *set1* und *set2* zurückliefern, das heißt ein Objekt a soll genau dann Element der zurückgegebenen Menge sein, wenn es *entweder* Element der Menge *set1*, *oder* Element der Menge *set2* ist. Das heißt insbesondere, dass Objekte, die sowohl Element der Menge *set1* als auch der Menge *set2* sind, nicht Element der zurückgegebenen Menge sein dürfen.

Die Argumente der Methode sollen dabei *nicht* verändert werden.

Sie können bei der Lösung dieser Aufgabe davon ausgehen, dass alle Objekte von Typ A sinnvolle Implementierungen der Methoden *hashCode* und *equals* zur Verfügung stellen. Außerdem können Sie davon ausgehen, dass *null* weder Element von *set1* noch von *set2* ist.

Lösungsvorschlag:

```
public Set⟨A⟩ symmetricDifference(Set⟨A⟩ set1, Set⟨A⟩ set2) {
  Set⟨A⟩ res = new HashSet⟨A⟩();
  for (A element : set1) {
    if (!set2.contains(element)) {
      res.add(element);
    }
  }
  for (A element : set2) {
    if (!set1.contains(element)) {
      res.add(element);
    }
  }
  return res;
}
public Set⟨A⟩ symmetricDifferenceAlt(Set⟨A⟩ set1, Set⟨A⟩ set2) {
  Set⟨A⟩ subRes1 = new HashSet⟨A⟩(set1);
  subRes1.removeAll(set2);
  Set⟨A⟩ subRes2 = new HashSet⟨A⟩(set2);
  subRes2.removeAll(set1);
  subRes1.addAll(subRes2);
  return subRes1;
}
```

RWTH Aachen **WS 2013/14**
Lehrgebiet Theoretische Informatik Blatt 7 Gruppe E
Frohn–Reidl–Rossmanith–Sánchez–Ströder

Minipräsenzübung zur Vorlesung Programmierung

Aufgabe 7 (10 Punkte)

Implementieren Sie eine Methode mit folgender Signatur:

public boolean *isSubMap*(*java.util.Map*⟨*A*, *B*⟩ *subMap*, *java.util.Map*⟨*A*, *B*⟩ *superMap*)

Diese Methode soll genau dann *true* zurückliefern, wenn für jedes Paar (k, v), für das *subMap.get*(k).*equals*(v) gilt, auch *superMap.get*(k).*equals*(v) gilt.

Die Argumente der Methode sollen dabei *nicht* verändert werden.

Sie können bei der Lösung dieser Aufgabe davon ausgehen, dass alle Objekte von Typ A oder B sinnvolle Implementierungen der Methoden *hashCode* und *equals* zur Verfügung stellen. Außerdem können Sie davon ausgehen, dass sowohl *subMap* als auch *superMap* den Wert *null* weder als key noch als value enthalten.

Lösungsvorschlag:

```
public boolean isSubmap(Map⟨A, B⟩ subMap, Map⟨A, B⟩ superMap) {
  for (Entry⟨A, B⟩ e : subMap.entrySet()) {
    A key = e.getKey();
    B value = e.getValue();
    if (!value.equals(superMap.get(key))) {
      return false;
    }
  }
  return true;
}
public boolean isSubmapAlt(Map⟨A, B⟩ subMap, Map⟨A, B⟩ superMap) {
  return superMap.entrySet().containsAll(subMap.entrySet());
}
```