SAARLAND UNIVERSITY
COMPUTER SCIENCE

Arbeitsgruppe Programmierung
Jun.-Prof. Dr. Sebastian Hack
Dipl.-Inform. Christoph Mallon

Programmierung 2 — Übungsblatt 5

Abgabe: **Dienstag, 2. Juni 2009, 11.45 Uhr, Geb. E1 3, Briefkasten im EG**
Lösung mit **Name**, **Matrikelnummer** und **Name des Tutors** beschriften!

Aufgabe 5.1: Optimieren in ererbten Klassen und Transparenz bei deren Benutzung

Implementieren Sie eine Klasse `Polynom`. Die Objekte dieser Klasse repräsentieren Polynomfunktionen in einer Variablen eines fixen, aber beliebigen Grades $n \in \mathbb{N}_0$. Polynome besitzen bekanntlich die Form:

$$f : \mathbb{R} \to \mathbb{R} \text{ mit } f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 \text{ mit } x, a_i \in \mathbb{R}, i = n, \ldots, 0$$

Die Klasse soll folgendes leisten:

1. Dem Konstruktor wird nur ein Array `coeffs` des Typs `double[]` übergeben, wobei `coeffs` die $n + 1$ Koeffizienten a_i des zu erstellenden Polynom-Objekts enthält. a_0 hat stets den Index 0.

   Die Polynom-Klasse hat zu gewährleisten, dass nur Unterklassen Polynom-Objekte erzeugen können.

2. Sie bietet eine Methode `public double eval(double x)`, welche den Wert des Polynoms an der Stelle x berechnet.

3. Weiter existieren folgende öffentliche Methoden:

   - `add()`, `sub()` und `mul()`, die jeweils ein Polynom als Parameter besitzen und ein neues Polynom als Ergebnis liefern, welches jeweils die Summe, die Differenz bzw. das Produkt der beiden Polynome darstellt.
   - `derive()`, welche die Ableitung $f'(x)$ als Ergebnis liefert.
   - `toString()` zur Darstellung des Polynoms als Zeichenkette, z.B. `f(x) = 4x^3 + 0.5x^2 - x + 1.5`.
   - `double[] roots(double start, int iterlimit)`, welche die Nullstelle(n) eines Polynoms bestimmt. Es soll eine leere Reihung zurückgegeben werden, falls keine Nullstelle existiert. Verwenden Sie das Newton-Iterationsverfahren zur näherungsweisen Bestimmung der Nullstellen:

     $$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

     Dieses iterative Verfahren kann nicht immer eine Nullstelle finden, selbst wenn Nullstellen existieren. Ob eine Nullstelle bzw. eine bestimmte Nullstelle (von mehreren) gefunden werden kann, hängt von einem geeigneten Startwert für die Iteration ab. Allgemein kann die Iteration entweder gegen eine Nullstelle konvergieren oder divergieren. Divergiert sie, hat man keinen Hinweis auf ein Nullstelle. Deshalb ist bei Aufruf dieser Methode noch ein Wert `iterlimit` anzugeben, welcher die Anzahl der Iterationen beschränkt. Um bei Iterationsabbruch zu entscheiden, ob vermutlich gegen eine Nullstelle konvergiert wurde, gegen Sie folgendermaßen vor: Falls $|f(x_{iterlimi})| < \epsilon$ mit ϵ nahe bei 0, dann geben Sie $x_{iterlimit}$ als Nullstelle zurück.

Erstellen Sie erbende Klassen:

Für Polynome vom Grad $n \leq 2$ können die Nullstellen direkt berechnet werden. Dies ist dann wesentlich effizienter als der iterative Ansatz. Schreiben Sie Klassen `Constant` (Grad 0), `Line` (Grad 1) und `Parabola` (Grad 2), welche von `Polynom` erben, und jeweils die Methode `roots()` der Polynom-Klasse überschreiben; die Nullstellen dieser speziellen Polynome werden in direkter Weise berechnet. Somit ignorieren diese Methoden die Parameter `start` und `iterlimit`.
Hinweis: Falls unendlich viele Nullstellen existieren, liefert `roots()` nur eine der Nullstellen zurück.

Verbergen Sie optimierte Realisierungen von Polynomen vor Nutzern:

Objekte, welche Polynome nutzen, sollen die verschiedenen Realisierungen eines Polynom-Objekts, die aus Optimierungsgründen eingeführt wurden, nicht wahrnehmen. Dazu wird eine spezielle Klassenmethode `create()` in `Polynom` eingeführt: Sie erzeugt implementierungsspezifische Polynom-Objekte in Abhängigkeit vom Polynomgrad, der bei der Erzeugung eines Polynom-Objekts gefordert ist. Der Rückgabetyp ist aber stets der Typ Polynom.

```
public static Polynom create(double ... v) {
  switch (v.length) {
    case 3:  return new Parabola(v);
    case 2:  return new Line(v);
    case 1:  return new Constant(v);
    default: return new Polynom(v);
  }
}
```

Hinweis: Beachten Sie, dass einige der oben genannten Operationen auf Polynomen (z.B. $f(x) \pm g(x)$, $f(x) * g(x)$, $f'(x)$) als Ergebnis neue Polynome erzeugen können, die einen anderen Grad besitzen als die Operanden-Polynome. Auch dann ist diese spezielle „Fabrik"-Methode Polynom.create einzusetzen!

Testwerte:

Welche Nullstellen liefern die folgenden Aufrufe?

- `Polynom p = Polynom.create(5); double[] r = p.roots(0,0);`
- `Polynom p = Polynom.create(0); double[] r = p.roots(0,0);`
- `Polynom p = Polynom.create(4,2); double[] r = p.roots(0,0);`
- `Polynom p = Polynom.create(-6,1,1); double[] r = p.roots(0,0);`
- `Polynom p = Polynom.create(-1, 0.5, 10.5, 33.33, 12, 34.05, 0.05);`
  `double[] r = p.roots(1000,16);`
- `Polynom p = Polynom.create(-1, 0.5, 10.5, 33.33, 12, 34.05, 0.05);`
  `double[] r = p.roots(-1000,16);`
- `Polynom p = Polynom.create(-1, 0.5, 10.5, 33.33, 12, 34.05, 0.05);`
  `double[] r = p.roots(-500,16);`

Aufgabe 5.2: Einsatz von HashMaps: Caching von vielen Polynom-Objekten

Für Polynome sind Ketten von Ableitungen möglich. Erzeugte abgeleitete Polynome sollen wiederbenutzt werden. Zu diesem Zweck unterhalten wir sie in einem Cache für Paare von Objekten, welche ein Polynom und sein Ableitung repräsentieren. Allerdings sollte ein Polynom-Cache (als Klasse) allgemeiner nutzbar sein: Jede Instantiierung sollte eine spezifische binäre Beziehung mit ihren zugehörigen Objekten unterhalten können. Die uns hier interessierende Ableitungsbeziehung ist nur ein Beispiel für die Nutzung eines solchen Polynom-Caches.

1. Implementieren Sie eine Klasse `PolynomCache`, um eine Sammlung von Paare an Polynom-Objekten zu verwalten. Diese Polynom-Objekte sind Instantiierungen der Klasse `Polynom` (und deren ererbten Klassen). Die eigentliche Sammlung an solchen Objektpaaren soll hierbei über eine Java `HashMap` realisiert werden.

2. Neben den üblichen Methoden, um Objekt-Paare im Polynom-Cache zu plazieren, zu entfernen und zu suchen, implementieren Sie eine Methode `Polynom[] getPolynomChain(Polynom p)`, die für ein Polynom `p` aus der zugehörigen HashMap die längste Folge $(p, p_2), (p_2, p_3), \ldots, (p_{n-1}, p_n)$ mit $(p_i, p_{i+1}) \in HashMap$ als Reihung $p, p_2, p_3, \ldots, p_{n-1}, p_n$ zurückliefert.

3. Damit Polynom-Objekte in einer Java HashMap eingesetzt werden können, müssen diese die Methoden `equals()` und `hashCode()`, die sie von `Object` erben, geeignet überschreiben. Dasselbe gilt für die von Polynom erbenden Klassen `Const`, `Line` und `Parabola`.

   Hinweis: Beachten Sie die nachfolgende Aufgabe.

Aufgabe 5.3: Überladen, Überschreiben und andere Fallen

Obgleich man auf den ersten Blick den Wert `true` erwarten könnte, geben die beiden folgenden Programme `false` aus. Warum? Korrigieren Sie die Programme.

1.
```
import java.util.*;
public class Name {
  private final String first, last;
  public Name(String first, String last) {
    this.first=first; this.last=last;
  }
  public boolean equals(Name n) {
    return n.first.equals(first) && n.last.equals(last);
  }
  public int hashCode() {
    return 31 * first.hashCode() + last.hashCode();
  }
  public static void main(String[] args) {
    Set<Name> s = new HashSet<Name>();
    s.add(new Name("Montgomery", "Scott"));
    System.out.println(s.contains(new Name("Montgomery", "Scott")));
  }
}
```

2.
```
import java.util.*;
public class Name {
  private final String first, last;
  public Name(String first, String last) {
    this.first=first; this.last=last;
  }
  public boolean equals(Object o) {
    if (!(o instanceof Name)) return false;
    Name n = (Name)o;
    return n.first.equals(first) && n.last.equals(last);
  }
  public static void main(String[] args) {
    Set<Name> s = new HashSet<Name>();
    s.add(new Name("Montgomery", "Scott"));
    System.out.println(s.contains(new Name("Montgomery", "Scott")));
  }
}
```