Aufgabe 11.1 (Ü) Überschreiben und Überladen von Methoden

Betrachten Sie folgenden Klassendeklarationen:

```
public class A {
    public void f() {
        System.out.println("bin Methode f in Klasse A");
    }
    public void g() {
        System.out.println("bin Methode g in Klasse A");
    }
}
public class B extends A {
    public void f() {
        System.out.println("bin Methode f in Klasse B");
    }
}
public class C {
    private A a;
    public C(A a) {
         this.a = a;
    }
    public void g(){
         System.out.println("bin Methode g in Klasse C");
          a.f();
    }
}
public class D {
    public void f(A a) {
        a.f();
        System.out.println("bin Methode f in Klasse D mit Typ A");
    }
    public void f(B b) {
        b.f();
        System.out.println("bin Methode f in Klasse D mit Typ B");
    }
}
public static void main (String[] s){
    A a0 = new A();
    a0.f();
    a0.g();
    A a1 = new B();
    a1.f();
    a1.g();

    C c0 = new C(a0);
    c0.g();
```

```
    C c1 = new C(a1);
    c1.g();

    D d = new D();
    d.f(a0);
    d.f(a1);
    d.f(new B());
}
```

Welche Ausgaben erwarten Sie für die jeweiligen Methodenaufrufe? Begründen Sie Ihre Antwort.

Lösungsvorschlag 11.1

a) zur Compilierzeit:
   betrachte die Signatur der Funktion: Funktionsname und Typ der Parameter (nur die Deklaration ist entscheidend)
b) zur Laufzeit:
   Entscheidung welche Funktion (welcher Klasse) aufgerufen wird

Folgende Ausgaben werden produziert:

- bin Methode f in Klasse A
- bin Methode g in Klasse A
- bin Methode f in Klasse B
- bin Methode g in Klasse A
- bin Methode g in Klasse C; bin Methode f in Klasse A
- bin Methode g in Klasse C; bin Methode f in Klasse B
- bin Methode f in Klasse A; bin Methode f in Klasse D mit Typ A
- bin Methode f in Klasse B; bin Methode f in Klasse D mit Typ A
- bin Methode f in Klasse B; bin Methode f in Klasse D mit Typ B

Aufgabe 11.2 (Ü) Termine

Termine haben haben einen Zeitpunkt, eine Beschreibung und finden an einem Ort statt. Man unterscheidet Termine, die

- einmalig,
- wiederholt alle n Tage (endlos) sowie
- wiederholt alle n Tage innerhalb einer festen Zeitspanne

stattfinden. Ziel dieser Aufgabe ist es, diese Hierarchie und die entsprechenden Gemeinsamkeiten durch Vererbung möglichst gut umzusetzen.

Zur Vereinfachung gehen wir davon aus, dass alle Zeitangaben (also auch Beginn und Ende von Zeitspannen) in Tagen nach Beginn Ihres Studiums (1. Oktober) erfolgen. Der `int`-Wert 6 repräsentiert also den 7. Oktober.

Programmieren Sie eine Klasse `Termin` sowie die davon abgeleitete Klassenhierarchie zur Darstellung einmaliger, sich endlos wiederholender sowie sich nur innerhalb einer vorgegebenen Zeitspanne wiederholender Termine.

Ausserdem soll die Klasse `Termin` eine Methode `int diff(int tag)` zur Verfügung stellen, die bestimmt, in wievielen Tagen relativ zum Zeitpunkt `tag` der entsprechende Termin zum nächsten Mal stattfindet. Im Falle vergangener, einmaliger Termine wird entsprechend eine

negative Zahl zurückgegeben. Für sich wiederholende Termine betrachten Sie die folgenden Beispiele:

- Ein sich endlos wiederholender Termin, der seit dem 25. Tag jede Woche wiederholt stattfindet, liefert für `t.diff(27)` die Zahl 5 zurück.
- Ein Termin, der sich in der Zeit vom 2. bis zum 32. Tag alle 3 Tage wiederholt $(2, 5, \ldots, 26, 29, 32)$, liefert für `t.diff(27)` die Zahl 2 zurück.
- Für denselben Termin, der also vom 2. bis zum 32. Tag alle 3 Tage stattfindet, liefert `t.diff(33)` entsprechend die Zahl -1 zurück.

Schreiben Sie eine Klasse `TerminTest`, in deren `main`-Methode Sie Ihre Klassen testen.

Lösungsvorschlag 11.2

```
public class Termin {

    private int tag;
    private String beschreibung;
    private String ort;

    public Termin(int tag, String beschreibung, String ort) {
        this.tag = tag;
        this.beschreibung = beschreibung;
        this.ort = ort;
    }

    public int diff(int tag) {
        return this.tag - tag;
    }

    public int getTag() {
        return tag;
    }

    public String getBeschreibung() {
        return beschreibung;
    }

    public String getOrt() {
        return ort;
    }

}

public class PeriodEwigTermin extends Termin {

    private int periode;

    public PeriodEwigTermin(int tag, int periode,
                            String beschreibung, String ort) {
        super(tag, beschreibung, ort);
        this.periode = periode;
    }

    public int getPeriode() {
        return periode;
```

```
    }

    @Override
    public int diff(int tag) {
        int terminAnfang = getTag();
        if (getTag() > terminAnfang) {
            return super.diff(tag);
        } else {
            while (terminAnfang - tag < 0) {
                terminAnfang = terminAnfang + periode;
            }
            return terminAnfang - tag;
        }
    }
}

public class PeriodTermin extends PeriodEwigTermin {

    private int ende;

    public PeriodTermin(int tag, int ende, int periode,
                        String beschreibung, String orte)
    {
        super(tag, periode, beschreibung, orte);
        this.ende = ende;
    }

    public int getEnde() {
        return ende;
    }

    @Override
    public int diff(int tag) {
        if (tag > ende)
            return ende - tag;
        else
            return super.diff(tag);
    }

}
```

Aufgabe 11.3 (H) Terminkalender

Führen Sie Aufgabe 11.2 fort in dem Sie die Klasse `Terminkalender` implementieren, mit deren Hilfe

- Termine eingetragen (`void fuegeTerminHinzu(Termin t)`),
- Termine eines Tages abgefragt (`Termin[] termineAm(int tag)`) und
- der relativ zu einem Tag `tag` zukünftig als nächstes stattfindende Termin bestimmt (`Termin naechsterTermin(int tag)`)

werden können. **Hinweis:** Zur Verwaltung der Liste der Termine können Sie die angepasste Listenimplementierung auf der Übungshomepage verwenden. Beachten Sie auch das Sie Objekte der Unterklassen in Listen/Arrays des Typen einer Oberklasse Speichern können. Dieses Konzept nennt man dynamisches Binden und wird in der nächsten Vorlesung behandelt. In unserem Fall können Sie Termine jeglichen Typs in einer Liste mit einträgen des Klassentyps `Termin` speichern.

Schreiben Sie eine Klasse `TerminkalenderTest`, in deren `main`-Methode Sie Ihre Klassen und die erwarteten Rückgabewerte testen.

Lösungsvorschlag 11.3

```
public class Terminkalender {

    TerminListe termine;

    public void fuegeTerminHinzu(Termin termin) {
        if (termine == null)
            termine = new TerminListe(termin);
        else
            termine = termine.add(termin);
    }

    public Termin[] termineAm(int tag) {
        if (termine == null)
            return new Termin[0];

        // Determine the number of appointments at the given day.
        int anzahl = 0;
        for (TerminListe tl = termine; tl != null; tl = tl.getNext()) {
            if (tl.getTermin().diff(tag) == 0)
                anzahl++;
        }

        // Create and fill an array containing the appointments at the
           given day.
        Termin[] result = new Termin[anzahl];
        int index = 0;
        for (TerminListe tl = termine; tl != null; tl = tl.getNext()) {
            if (tl.getTermin().diff(tag) == 0) {
                result[index] = tl.getTermin();
                index++;
            }
        }
        return result;
    }

    public Termin naechsterTermin(int tag) {
        if (termine == null)
            return null;

        Termin nt = null;
        int minTage = Integer.MAX_VALUE;
        for (TerminListe tl = termine; tl != null; tl = tl.getNext()) {
            if (tl.getTermin().diff(tag) < minTage) {
                nt = tl.getTermin();
                minTage = tl.getTermin().diff(tag);
            }
        }
        return nt;
    }

}
```

```
public class TerminkalenderTest {

    public static void main(String[] args) {
        Terminkalender tk = new Terminkalender();

        Termin event1 = new Termin(1, "Normaler Termin am 2. Tag", "Ort
            A");
        Termin event2 = new Termin(3, "Normaler Termin am 4. Tag", "Ort
            B");
        PeriodEwigTermin event3 = new PeriodEwigTermin(3, 7, "Wdh. von
            3 alle 7 Tage", "Ort C");
        PeriodTermin event4 = new PeriodTermin(3, 10, 2, "Wdh. von 3
            bis 10 alle 2 Tage", "Ort D");

        tk.fuegeTerminHinzu(event1);
        tk.fuegeTerminHinzu(event2);
        tk.fuegeTerminHinzu(event3);
        tk.fuegeTerminHinzu(event4);

        Termin[] t1 = tk.termineAm(1);
        assert(t1.length == 1 && t1[0] == event1);

        Termin[] t2 = tk.termineAm(3);
        assert(t2.length == 3 && t2[0] == event2 && t2[1] == event3 &&
            t2[2] == event4);

        Termin[] t3 = tk.termineAm(17);
        assert(t3.length == 1 && t3[0] == event3);

        Termin nt = tk.naechsterTermin(2);
        assert(nt == event2 || nt == event3 || nt == event4);

        assert(event1.diff(2) == -1);
        assert(event2.diff(2) == 1);
        assert(event3.diff(9) == 1);
        assert(event4.diff(11) == -1);
    }

}
```