INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Prof. Dr. Dr. h.c. M. Broy
Dr. H. Ehler, S. Wagner

Lösungsblatt 10
18. Januar 2007

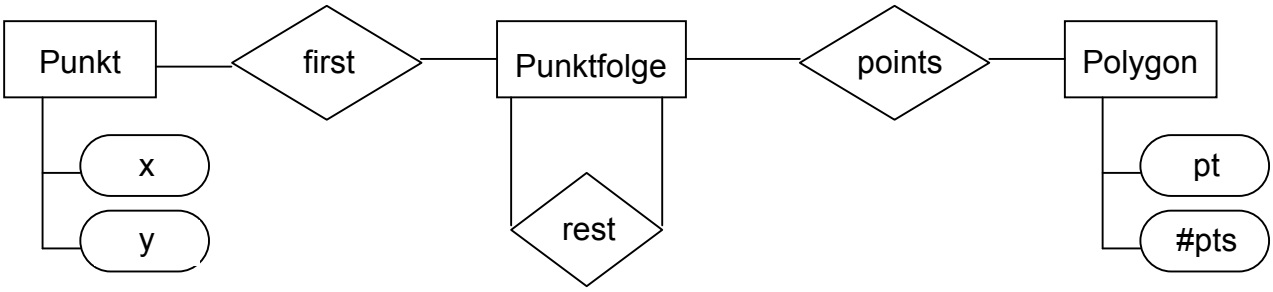
Übungen zu Softwaretechnik

Aufgabe 17 – Architekturbeschreibung

In dieser Aufgabe soll die Architektur für ein einfaches „Softwaresystem“ erstellt werden, das den Schwerpunkt eines einzugebenden Polygons berechnet und das Ergebnis graphisch ausgibt.

a) Bestimmen und beschreiben Sie hierzu folgende Bestandteile der Architektur:

1. *Datenmodell*



2. *Strukturierung in Komponenten*
3. *Spezifikation der Methoden nach „design by contract“*

Punkt

```
x: real; y: real; sort Punkt;

method create (): Punkt;
     pre: true;
     post: (neuer p: Punkt existiert) ∧ (getX(p)=getY(p)=0)

method dispose (rw p: Punkt);
     pre: p existiert;     post: p existiert nicht mehr

method getX (r p: Punkt): real;
     pre: p existiert;     post: getX(p)=p.x

method getY (r p: Punkt): real;
     pre: p existiert;     post: getY(p)=p.y

method setX (rw p: Punkt; r z: real);
     pre: p existiert;     post: (getX(p)=z) ∧ (getY(p)=getY(p'))

method setY (rw p: Punkt; r z: real);
     pre: p existiert;     post: (getX(p)= getX(p')) ∧ (getY(p)=z)
```

Polygon

```
pt: int; #pts: int; sort Polygon;

method create (): Polygon;
      pre: true;
      post: (neues pl: Polygon existiert) ∧ (pl.pt=pl.#pts=0)

method dispose (rw pl: Polygon);
      pre: pl existiert;    post: pl existiert nicht mehr

method isempty (r pl: Polygon): boolean;
      pre: pl existiert;    post: isempty(pl) ⇔ (pl.#pts=0)

method add (rw pl: Polygon; r p: Punkt);
      pre: pl existiert; p existiert;
      post: (pl.pt=pl.pt') ∧ (pl.#pts=pl.#pts'+1) ∧ (pl=pl'∘<p>)

method first (rw pl: Polygon): Punkt;
      pre: pl existiert ∧ ¬isempty(pl);
      post: (pl=pl') ∧ (pl.pt=1) ∧ (first(pl)=pl[1])

method next (rw pl: Polygon; w p: Punkt): boolean;
      pre: pl existiert ∧ ¬isempty(pl)
      post: (pl=pl') ∧
            (next(pl) ⇒
              (pl.pt'<pl.#pts) ∧ (pl.pt=pl.pt'+1) ∧ (p=pl[ pl.pt]))∧
            (¬next(pl) ⇒
              (pl.pt=pl.#pts) ∧ (pl.pt=pl.pt'))

method center (rw pl: Polygon): Punkt;
      pre: pl existiert ∧ ¬isempty(pl) ∧ intersectionsfree(pl);
      post: center(pl) existiert und ist Schwerpunkt von pl

method intersectionsfree (r pl: Polygon): boolean;
      pre: pl existiert;
      post: intersectionsfree(pl) ⇔ pl überschneidungsfrei
```

Polygoneingabe

```
method read (): Polygon;
      pre: true;
      post: read() existiert und ist Polygon aus einer über die
            Tastatur eingelesenen Punktefolge
```

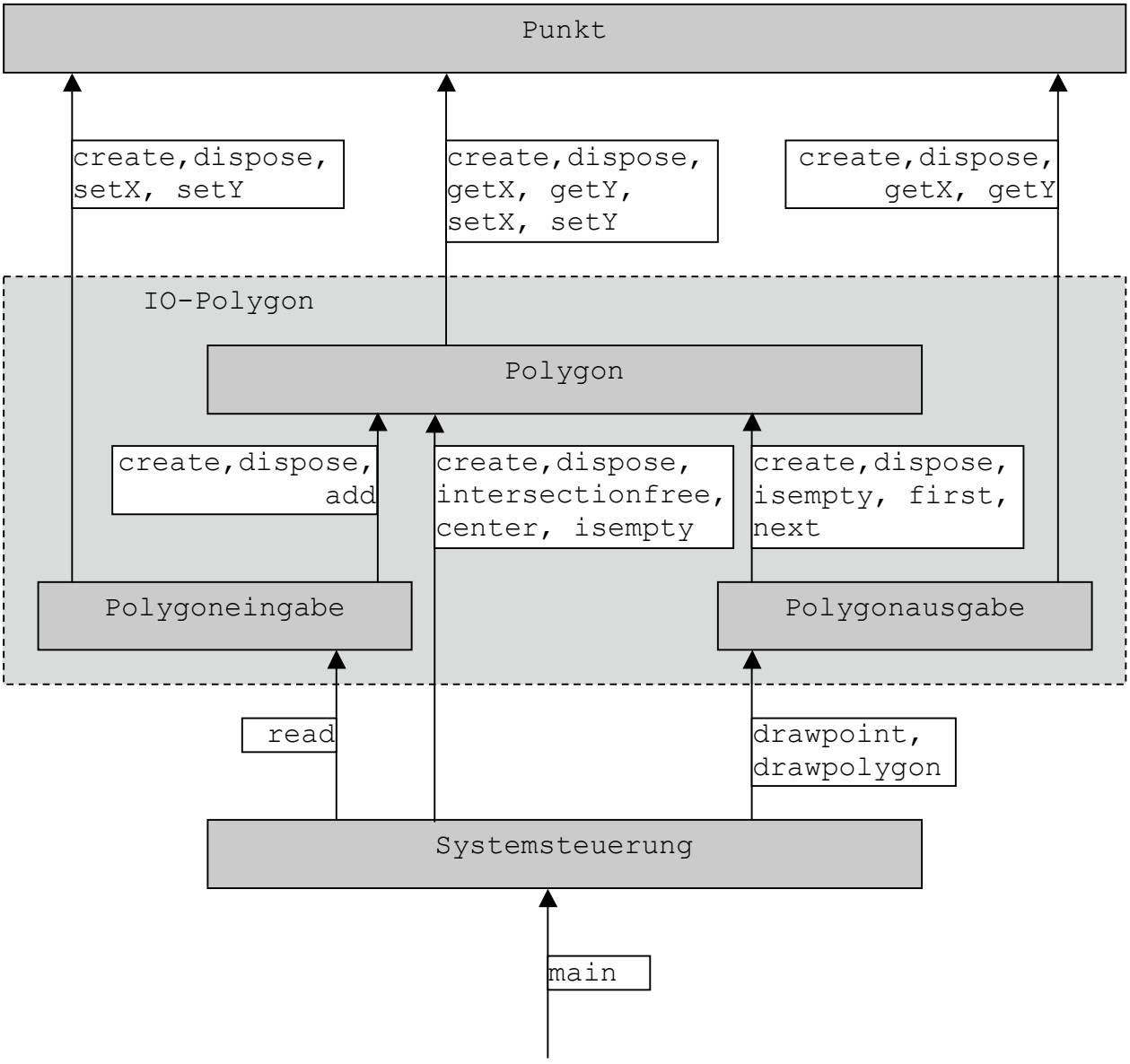
Polygonausgabe

```
method drawpoint (r p: Punkt);
      pre: p existiert;
      post: Ausgabe von p gemäß Zeichenspezifikation auf
            Standardausgabegerät

method drawpolygon (rw pl: Polygon);
      pre: pl existiert;
      post: Ausgabe von pl gemäß Zeichenspezifikation auf
            Standardausgabegerät
```

Systemsteuerung

```
p: Punkt; pl: Polygon;

method main ();
      pre: true;
      post: (pl=read()) ∧ (pl existiert ⇒
            (instersectionfree(pl) ∧ ¬isempty(pl) ⇒
                  (p=center(pl)) ∧
                  Ausgabe von pl gemäß Zeichenspezifikation auf
                  Standardausgabegerät) ∧
            (¬instersectionfree(pl) ∨ isempty(pl) ⇒
                  Ausgabe von Fehlermeldung auf Standardausgabegerät))
```

4. Aufrufbeziehung zwischen den Komponenten



5. Invarianten (Integritätsbedingungen)

In diesem Beispiel sind keine Invarianten nötig.

b) Extrahieren Sie aus der Architekturspezifikation beispielhaft ausgewählte Komponentenspezifikationen.

Als Beispiel wird der Modul Polygon extrahiert:

```
modul Polygon

import
     aus modul Punkt importiere
          sort Punkt,
          method create, dispose, getX, getY, setX, setY

export
     sort Punkt, Polygon,
     method create, dispose, isempty, add, first, next,
            intersectionfree, center

spec

pt: int; #pts: int; sort Polygon;

method create (): Polygon;
     pre: true;
     post: (neues pl: Polygon existiert) ∧ (pl.pt=pl.#pts=0)

method dispose (rw pl: Polygon);
     pre: pl existiert;   post: pl existiert nicht mehr

method isempty (r pl: Polygon): boolean;
     pre: pl existiert;   post: isempty(pl) ⇔ (pl.#pts=0)

method add (rw pl: Polygon; r p: Punkt);
     pre: pl existiert; p existiert;
     post: (pl.pt=pl.pt') ∧ (pl.#pts=pl.#pts'+1) ∧ (pl=sequ pl'∘<p>)

method first (rw pl: Polygon): Punkt;
     pre: pl existiert ∧ ¬isempty(pl);
     post: (pl=sequ pl') ∧ (pl.pt=1) ∧ (first(pl)=pl[1])

method next (rw pl: Polygon; w p: Punkt): boolean;
     pre: pl existiert ∧ ¬isempty(pl)
     post: (pl=sequ pl') ∧
           (next(pl) ⇒
             (pl.pt'<pl.#pts) ∧ (pl.pt=pl.pt'+1) ∧ (p=pl[ pl.pt]))∧
           (¬next(pl) ⇒
             (pl.pt=pl.#pts) ∧ (pl.pt=pl.pt'))

method center (rw pl: Polygon): Punkt;
     pre: pl existiert ∧ ¬isempty(pl) ∧ intersectionsfree(pl);
     post: center(pl) existiert und ist Schwerpunkt von pl

method intersectionsfree (r pl: Polygon): boolean;
     pre: pl existiert;
     post: intersectionsfree(pl) ⇔ pl überschneidungsfrei

end module
```