Fragen, Antworten, Kommentare zur aktuellen Vorlesung

Info: Die Projektwoche ist für Erstsemester*innen ein sehr guter Reflexionszeitpunkt, ob sie in den jeweiligen Veranstaltungen „brauchbar“ mitkommen. Sollte dies nicht der Fall sein oder Sie der Typ sein „ich hab ja die Unterlagen, das hole ich schnell nach“ sollten Sie über einen Beratungstermin bei der Studienberatung, dem Studiendekan oder auch bei mir ernsthaft nachdenken.

Wollen Sie in der Projektwochenzeit über Programmierung reden, bleiben z. B. bei einem Problem hängen, können Sie auch gerne mit mir einen Zoom-Termin vereinbaren.

Ausflug: Bei der Praktikumsaufgabe zur Erstellung der Klassen Punkt und Kreis stellt sich die Frage, wer erstellt eigentlich wann Objekte. Für Personen am absoluten Anfang der Programmiererfahrungen ist die nachfolgende Überlegung ein Ausflug in deutlich später folgende Themen, sie sollten aber mindestens den entstehenden Code verstehen. Der Startpunkt der Diskussion ist der folgende Konstruktor.

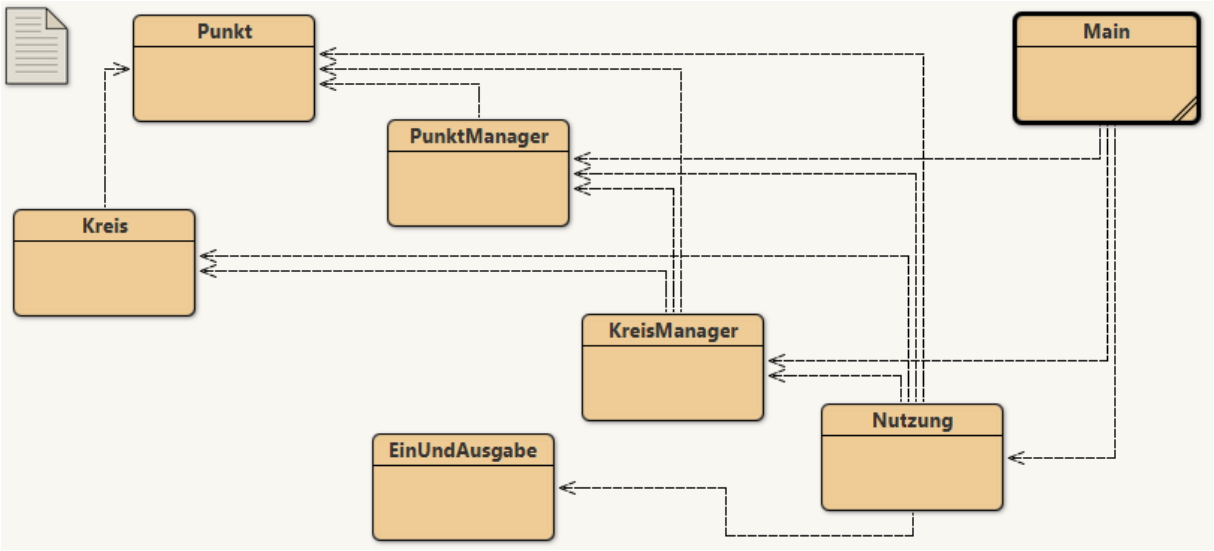
```
class Kreis {
  Punkt aufhaengepunkt; // Namen von Objektvariablen koennen anders sein
  int radius;

  Kreis(int x, int y, int radius){ // funktional ok, eher weglassen
    this.aufhaengepunkt = new Punkt(x,y);
    this.radius = radius;
  }
```

Generell funktioniert der Konstruktor und ist für Personen am absoluten Anfang ein sinnvolles Ergebnis. Allerdings ist die Grundlage der Objektorientierung das Zusammenspiel von Objekten, d. h. Objekte nutzen andere Objekte. Deshalb ist der wichtige Konstruktor der folgende:

```
  Kreis(Punkt aufhaengepunkt, int radius){ // ueblicher Konstruktor
    this.aufhaengepunkt = aufhaengepunkt;
    this.radius = radius;
  }
```

Natürlich geht es ganz ohne Konstruktoren, es werden dann set-Methoden genutzt.

Neben den Experimenten, die in der Veranstaltung im Mittelpunkt stehen und in denen an beliebigen Stellen Objekte erstellt werden, stellt sich die Frage, wie es später in der Praxis aussieht. Das zentrale Ziel in der Praxis ist immer wartbarer und erweiterbarer Code. Ein Teilziel davon ist die Möglichkeit kleine funktionale Veränderungen an Programmen vornehmen zu können ohne viele Klassen bearbeiten zu müssen. Findet dann eine Objekterstellung einer Klasse, also Aufruf von new, an vielen Stellen statt und soll das Verhalten bei der Objekterstellung angepasst werden, kann dies schnell sehr aufwändig werden. Ein fiktives Beispiel ist, dass bei der Erstellung eines Punktes geklärt werden soll, ob der Aufrufer überhaupt die Rechte dazu hat. Konkret werden diese Rechte wieder mit einem Objekt verwaltet, dass bei Erstellung genutzt werden soll. Eine nullte Idee könnte sein, diese Überprüfung in den Konstruktor einzubauen. Da aber Klassen nur eine Kernaufgabe haben sollen, ist das ein falscher Ansatz. Ein besserer Ansatz ist es, die Erstellung von Objekten jeweils an einer Stelle, genauer in er Klasse zu fokussieren. Hierzu werden gerne Verwaltungs- oder engl. Manager-Klassen genutzt.



Dies soll anhand eines Beispiels erklärt werden. Das zugehörige BlueJ-Diagramm wie folgt aus, nur die Klassen Punkt und Kreis werden als bekannt vorausgesetzt.

Die Erzeugung von Punkt-Objekten übernimmt der PunktManager. Jedes Objekt, das einen Punkt haben will, muss diesen Manager nutzen. Damit gibt es später nur eine Stelle, an der Punkt-Objekte erzeugt werden.

```
class PunktManager {

  Punkt erzeugePunkt(int x, int y){
    return new Punkt(x, y);
  }
}
```

Ähnlich arbeitet der KreisManager, der allgemein Kreise verwalten und damit auch ihre Erzeugung übernehmen wird. Hier ist es diskutabel, ob zur Objekterzeugung bereits ein Punkt existieren muss oder dieser hier erzeugt werden kann. Die zweite Variante kann „zur Bequemlichkeit“ auch angeboten werden. Es ist am Code erkennbar, dass ein Manager einen anderen nutzen kann.

```
class KreisManager {
  PunktManager punktmanager;

  KreisManager(PunktManager pm){
    this.punktmanager = pm;
  }

  Kreis erzeugeKreis(int x, int y, int radius){ // evtl. aus Bequemlichkeit
    Punkt tmp = this.punktmanager.erzeugePunkt(x, y);
    return new Kreis(tmp, radius);
  }

  Kreis erzeugeKreis(Punkt punkt, int radius){ // der wichtige
    return new Kreis(punkt, radius);
  }
}
```

Konsequent stellt sich die Frage, wer die Manager erzeugt. Im einfachsten und oft vorkommenden Fall gibt es eine zentrale Klasse mit einer zentralen Methode, in der die Manager-Objekt erzeugt und an alle nutzenden Klassen, auch zur späteren Weiterverteilung, übergeben werden. Dies zeigt die folgende Beispielklasse.

```
class Main {
  void main(){
    PunktManager punktmanager = new PunktManager();
    KreisManager kreismanager = new KreisManager(punktmanager);
    Nutzung nutzung = new Nutzung(punktmanager, kreismanager);
    nutzung.nutzen();
  }
}
```

Die nutzenden Klassen bekommen die Manager übergeben, werden sie niemals selbst erzeugen und können die Manager über die angebotenen Methoden nutzen. Dieser Ansatz wird auch als „Dependency Injection“ bezeichnet, benötigte Objekte werden über Konstruktoren oder set-Methoden übergeben. Eine Beispielnutzung kann wie folgt aussehen, die erwähnte Klasse ToStringer von der Veranstaltungsseitewird hier als eventuell irritierende Spielerei mal gezeigt und spielt bei den eigentlichen Überlegungen keine Rolle.

```
import gsdet.toStringer.ToStringActivator; // loeschen, wenn nicht in kleukersSEU

class Nutzung {
  PunktManager punktmanager;
  KreisManager kreismanager;

  Nutzung(PunktManager pm, KreisManager km){
    this.punktmanager = pm;
    this.kreismanager = km;
  }

  void nutzen(){
    Punkt pu = this.punktmanager.erzeugePunkt(26, 2);
    Kreis kr = this.kreismanager.erzeugeKreis(pu, 20);
    EinUndAusgabe io = new EinUndAusgabe();
    ToStringActivator.activate(); // loeschen, wenn nicht in kleukersSEU
    io.ausgeben(pu + "\n");
    io.ausgeben(kr + "\n");
  }
}
```

Die Ausgabe des Programms sieht wie folgt aus.

```
<Punkt x=26,y=2>
<Kreis aufhaengepunkt=<Punkt x=26,y=2>,radius=20>
```