# Lisaülesanne 5

Iteraatori näitena vaatleme siin *PointIterable* liidest, mis tagastab iteraatori üle punktide.

*PointIterable* kasutamise näide Pythonis:

for p in point\_iterable: print(p.x(), p.y())

Pythoni klass *PlanetarySystem* on *PointIterable*, kuna selle elemendid on punktid ja klassi *list* alamklassina on see itereeritav objekt.

Java klass *PlanetarySystem* ei ole automaatselt *Iterable<Point>*, vaid seda on vaja täiendada, realiseerides *Iterable* liidese (implements Iterable<T>) ja kirjutades üle meetodi *iterator()*:

 @Override

 **public** Iterator<T> iterator() {

 // **TODO** Auto-generated method stub

 **return** elemendid.iterator();

 }

Seejärel saame me planeedisüsteemi kasutada punktihulka eeldavates meetodites:

**public** **void** drawPoints(Iterable<? **extends** Point> pi) {

 **for** (Point p: pi) {

….

NB! Loogilisena tunduv *Iterable<Point>* siin Java eripärade tõttu ei tööta, sest sel juhul ei saa *Pointi* alamklasse (*Iterable<PointSimulationElement>* jms.) argumendina kasutada.

1. Kirjeldada UML diagrammidega disain, kus on olemas klass *PointDrawer*, mis joonistab *PointIterable* liidest realiseeriva punktihulga ekraanile. Lisada disaini *PlanetarySystem* ja *SierpinskiTriangle* (vt loengu Ühendi(Composite) osa) klassid sel viisil, et *PointDraweril* on võimalik vastavaid punktihulki (planeedid, Sierpinski kolmnurga viimane tase) joonistada (*PlanetarySystem* ja *SierpinskiTriangle* on tüüpi *Iterable<Point>*) ja muudatused olemasolevas süsteemis oleksid minimaalsed.
2. Realiseerida see disain mingis OO keeles, kasutades vastava keele Iteraatori/Itereeritava liidest/liideseid.