W08B01 - HPC Polymorphism

Hausaufgabe Easy

Points: 6 of 6 Bonus Assessment: automatic

Submission due: 7 months ago

Complaint due: 7 months ago

100% (8 months ago) GRADED

Recent results:

66.67% (8 months ago)	83.33% (8 months ago)	83.33% (8 months ago)	100% (8 months ago)	100% (8 months ago)
GRADED	GRADED	GRADED	GRADED	GRADED

Show all results

Tasks:

High Performance Computing Polymorphism

Bonusaufgabe

Die ist eine Bonusaufgabe. Das heißt, die in dieser Aufgabe erreichten Punkte werden am Ende auf deine Gesamtpunktzahl draufgerechnet, die 6 in dieser Aufgabe erreichbaren Punkte sind aber nicht Teil der 250 im gesamten Kurs erreichbaren Punkte. Die Aufgabe ist also ein freiwilliger Zusatz, mit dem du dir ein Paar kostenlose Punkte dazuverdienen kannst.

Mit Punkten aus Bonusaufgaben kann allerdings *nicht* bestanden werden. D.h. wenn du z.B. am Ende mit den in den H-Aufgaben erreichten Punkten 2 Punkte unterhalb der Bestehensgrenze liegst und zusätzlich noch in dieser Aufgabe 4 Punkte erreicht hast, hast du dennoch nicht bestanden. Zudem zählen die in dieser Aufgabe erreichbaren Punkte auch nicht zu den 40%, die man benötigt, um Woche 08 zu "bestehen" (siehe "Vorstellung des PGdP Foliensatz" auf Moodle - Folie 11).

Die Aufgabe

Die Pinguine wollen ganz hoch hinaus. Um Intel, AMD und Co. Konkurrenz zu machen, wollen sie jetzt eigene Rechner entwickeln. Sie haben in Java bereits begonnen, eine Hierarchie zu entwickeln. Ein paar kleine Touch Ups sind aber noch notwendig. Du übernimmst das natürlich.

Achtung! In dieser Aufgabe darfst du das vorgegebene Template nur an den Markierten Stellen bearbeiten. Editierst du eine verbotene Stelle, verlierst du automatisch alle Punkte (Die Tests zeigen dir direkt eine Fehlermeldung an!). Halte dich also strikt daran. Die Stellen sind jeweils mit Kommentaren der entsprechenden Teilaufgabe markiert (`// TASK X (start)` und `// TASK X (end)`). Verändere auch diese Kommentare nicht.
Hinweis: Solltest du in deiner IDE Autoformatierung eingestellt haben, sorge bitte dafür, dass die Autoformatierung beim Speichern keine Veränderungen am Template vornimmt. Vor allem nicht an expliziten Casts die *IntelliJ* als *redundant* einstuft! *Hinweis:* Wie immer enthalten Strings in de Angabe zur besseren Lesbarkeit ␣. Wenn nötig sollen diese wie immer durch Leerzeichen im Programm ersetzt werden.

Tipps:

- Die Aufgabe selbst ist nicht allzu umfangreich, beschäftigt sich allerdings etwas tiefgehender mit dem Thema "Polymorphie", was im PGdP sonst nicht in der Tiefe abgefragt wird (das bezieht sich nicht auf die EidI-Klausur ;-)). Die im Template bereitgestellte Klassenhierarchie kann jedoch auf den ersten Blick ziemlich verwirrend aussehen. Es macht daher Sinn, sich die gesamte Hierarchie mitsamt Methoden in einem UML-Diagramm aufzuzeichnen und sich zu überlegen, welche Methoden in welcher Klasse jeweils welche anderen Methoden überschreiben. Stay calm, keep a cool head! Die Lösungen sind alle recht kurz und wenn man erstmal den Überblick über die Hierarchie hat und sich die Prinzipien der Polymorphie und des Aufrufs überschriebener Methoden ins Gedächtnis gerufen hat, nicht allzu schwierig zu erhalten.
- Es macht auch Sinn, sich nochmal die Folien von Woche 07 (und 08) durchzuschauen.
- Es gibt für diese Aufgabe keine Hidden Tests. Alles Feedback/alle Punkte, die du nach der Deadline erhältst, siehst du bereits vor der Deadline.

1. **A Computer and nothing else** 9 of 9 tests passing
   Implementiere die Methode `getComputer()`. Die Methode soll eine Instanz eines `Computer`s zurückgeben, die weder in ein anderes Interface, das `Computer` erweitert (z.B. `ComputeCluster`), noch in eine Klasse, die das Interface `Computer` implementiert (z.B. `Master`) gecastet werden kann. Wenn au dieser Instanz die Methode `connectByCopper` aufgerufen wird, soll diese Methode folgenden String ausgeben:
   "`Computer␣connected␣to␣Computer␣by␣copper.`")

2. **A special SuperMUC** 4 of 4 tests passing
   Implementiere die Methode `getSpecialSuperMUC(Computer c)`. Die Methode soll eine spezielle Instanz eines `SuperMUC`s zurückgeben. Diese spezielle Instanz soll bei dem Aufruf der Methode `connectByFiber` mit einem `Monitor<Master>` als Argument zunächst folgende Ausgabe tätigen:
   "`Special␣SuperMUC␣connected␣to␣Monitor␣by␣fiber.`" und anschließend soll beim Aufruf der Methode des speziellen `SuperMUC`s zusätzlich noch der `Computer c` (Argument der Methode `getSpecialSuperMUC(Computer c)`) mit sich selbst durch ein Kupferkabel verbunden werden (`connectByCopper`). (Sinnhaftigkeit? Wir sind pro Inklusion, haben also auch für sehr spezielle `SuperMUC`s einen Platz in unseren Herzen.)

3. **A disguised FPGA** 3 of 3 tests passing
   Implementiere die Methode `getFPGAasASIC()`. Die Methode soll eine Instanz eines `ASIC`s zurückgeben, die in einen `FPGA` gecastet werden kann.

4. **Ambiguous connection** 3 of 3 tests passing
   Wir möchten das Verhalten von `ASIC`s und `ASICBoard`s spezifizieren. Probiere in deiner IDE folgende Befehle aus. Wir initialisieren zwei `ASIC`s: `final ASIC<ComputeCluster<Computer>> a = new ASIC<>(); final ASICBoard<Computer> ab = new ASICBoard<>(null);`. Führen wir folgendes Statement aus: `ab.connectByFiber(ab, a);` erwarten wir die Zeile "`ASICBoard_connected_to_ASICBoard_and_ASIC_by_fiber.`". Füge eine neue Method zu `ASIC` hinzu, sodass der Aufruf `ab.connectByFiber(a, ab)` "`ASIC_connected_to_ASIC_and_ASICBoard_by_fiber.`" ausgibt und zusätzlich der Aufruf `ab.connectByFiber(ab, ab)` nicht kompiliert werden kann (In der IDE soll dir bei diesem Aufruf direkt ein `The Method [...] is ambiguous for the type HPCPoly.ASICBoard<HPCPoly.Computer>` angezeigt werden).

5. **Breaking infinity** 5 of 5 tests passing
   Sieh dir die folgenden Code-Ausschnitte und deren Verhalten im Template an:

```
final Master m = new Master();
final SuperMUC s = new SuperMUC();
s.connectByCopper(s);
// OUT: SuperMUC connected to SuperMUC by copper.
// OUT: Monitor connected to Computer by copper.
```

```
final Master m = new Master();
final SuperMUC s = new SuperMUC();
s.connectByCopper(m);
// OUT: Master connected to Master by copper.
// OUT: Master connected to Master by copper.
// OUT: Master connected to Master by copper.
// ...
// StackOverflowError
```

```
final Master m = new Master();
final SuperMUC s = new SuperMUC();
m.connectByCopper(s);
// OUT: Master connected to Master by copper.
// OUT: Master connected to Master by copper.
// OUT: Master connected to Master by copper.
// ...
// StackOverflowError
```

```
final Master m = new Master();
final SuperMUC s = new SuperMUC();
m.connectByCopper(m);
// OUT: Master connected to Master by copper.
// OUT: Master connected to Master by copper.
// OUT: Master connected to Master by copper.
// ...
// StackOverflowError
```

Wie du siehst führen Code-Snippet 2-4 zu einem StackOverflow. Füge nun genau eine Methode hinzu, sodass dieselben Beispiele zu folgendem Verhalten führen:

```
final Master m = new Master();
final SuperMUC s = new SuperMUC();
s.connectByCopper(s);
// OUT: SuperMUC connected to SuperMUC by copper.
// OUT: Monitor connected to Computer by copper.
```

```
final Master m = new Master();
final SuperMUC s = new SuperMUC();
s.connectByCopper(m);
// OUT: SuperMUC connected to Master by copper.
```

```java
final Master m = new Master();
final SuperMUC s = new SuperMUC();
m.connectByCopper(s);
// OUT: Master connected to Master by copper.
// OUT: SuperMUC connected to Master by copper.
```

```java
final Master m = new Master();
final SuperMUC s = new SuperMUC();
m.connectByCopper(m);
// OUT: Master connected to Master by copper.
// OUT: Master connected to Master by copper.
// OUT: Master connected to Master by copper.
// ...
// StackOverflowError
```

Bei 1 & 4 bleibt das Verhalten unverändert. Wie in allen anderen Methoden demonstiert darf auch diese Methode nur eine einzige Zeile auf der Konsole ausgeben. *Hinweis:* Nimm dir die Zeit, die Beispiele genau zu lesen und achte auf das Schema der Ausgaben: "<Klassen-Name>`_connected_to_`<Argument-Name>`_by_`<Methoden-Name>`.`"

6. ✓ **There will be no Exceptions** 2 of 2 tests passing
   Angenommen `fpga` ist ein `FPGA`. Ruft man auf diesem `FPGA` die folgenden beiden Methoden auf: `fpga.set(null); fpga.connectByCopper(fpga)`, wird im Template eine Zeile ausgegeben ("`FPGA_connected_to_Computer_by_copper.`") und danach eine `NullPointerException` geworfen. Deine Aufgabe ist es, eine einzige Methode zur Klasse `FPGA` hinzuzufügen, sodass das `FPGA` mit einem beliebigen anderen `FPGA` mit Kupferkabeln verwunden wird (Ausgabe: "`FPGA_connected_to_FPGA_by_copper.`"). Anschließend soll diese Methode auch noch das `ComputeCluster t` der Instanz des `FPGA`s, auf dem die Methode `connectByCopper` aufgerufen wurde, mit dem übergebenen `FPGA` mit einem Kupferkabel verbinden (das erfordert erneut einen Methodenaufruf).

Viel Erfolg!

Exercise details	
Release date:	Dec 8, 2022 18:30
Submission due:	Jan 8, 2023 18:00
Complaint due:	Jan 15, 2023 18:00

Every student is allowed to complain once per exercise. In total 1000 complaints are possible in this course. You still have **998** complaints left.