PGR112 – Step 8: Abstract classes and HashMap

Object Oriented Programming

Bogdan Marculescu/ bogdan.marculescu@kristiania.no

Agenda

- Aggregation
- Abstract classes and methods
- HashMap

Inheritance – brief recap

- How to obtain inheritance through the use of extends.
- We saw how subclass can inherit from superclass (parent- or base class).
  - Reuse of code
- Use of super (designers and methods).
- We looked at overriding methods (and rules for this).

Is-a

- We should use inheritance with caution, and it is only relevant if the classes act in an "is-a" relationship.
  - The example from the weekly assignments (Circle is a Shape)
  - Cat is an Animal
  - Banana is a Fruit
- Inheritance provides a strong link between two classes.

Aggregation

- We can also achieve code reuse by using aggregation. Unlike inheritance, aggregation represents a "has-a" relationship.
- Let's take an example from beginnersbook.

Composition

- A specialized and strong variant of aggregation.
- Can be formulated as a "part of" relationship.
- Examples:
- Heart - human
- Engine - car
- We will not emphasize composition in this topic, but relate to aggregation in general.
- Here you can read more about the difference.

Abstraction

- The essence of abstraction is preserving information that is relevant in a given context, and forgetting information that is irrelevant in that context. (John V. Guttag)
- We want to hide implementation details.
- We focus on what an object does, not how it does it.
- We can perform abstraction using two mechanisms in Java:
  - Interface (next week)
  - Abstract classes (this week)

Abstract classes

- Abstract classes thus hide implementation details, but still say what the object does.
- We declare an abstract class with the reserved word *abstract*.
- Ex: public abstract class MyClass {…}

Some rules for abstract classes

- Can have abstract and concrete (i.e. Non-abstract) methods.
- It is used as a parent class. Sub-classes then inherit the abstract class, and make sure to implement (by override) the required behaviour.
- Abstract classes cannot be instantiated: ~~new MyClass~~

Abstract methods

- Declared as abstract and has no implementation (method body).
- Example: public abstract void myBeautifulAbstractMethod ();

Demo – abstract classes and methods

- Our example will be based on the one from w3schools.

Implementing abstract methods

- The subclasses thus have to ensure that the abstract methods in the parent class are implemented. But do they HAVE to do it?
- No☺ But if they do not, then the class must be abstract. Implementation details are still missing.
- Let's demonstrate…

HashMap

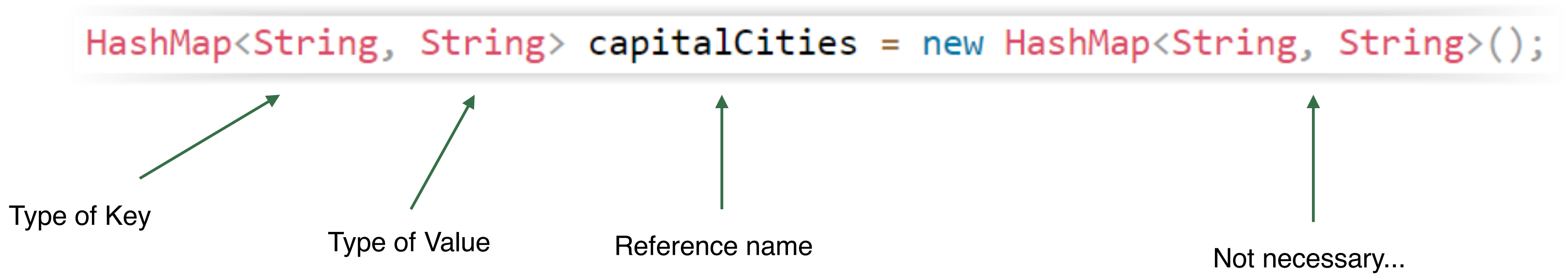
- Using ArrayList has some downsides.
- If we want to retrieve one specific object (without knowing the index), then it's a bit cumbersome.
- Let's look at an alternative: HashMap…

HashMap

- In a HashMap we can enter objects and retrieve them based on a key.
- We need to specify the type of keys and the type of objects. These can be different.
- Let's take an example from w3schools…

HashMap

- Create:



```
HashMap<String, String> capitalCities = new HashMap<String, String>();
```

Using HashMap

```
capitalCities.put("England", "London");
```

```
capitalCities.get("England");
```

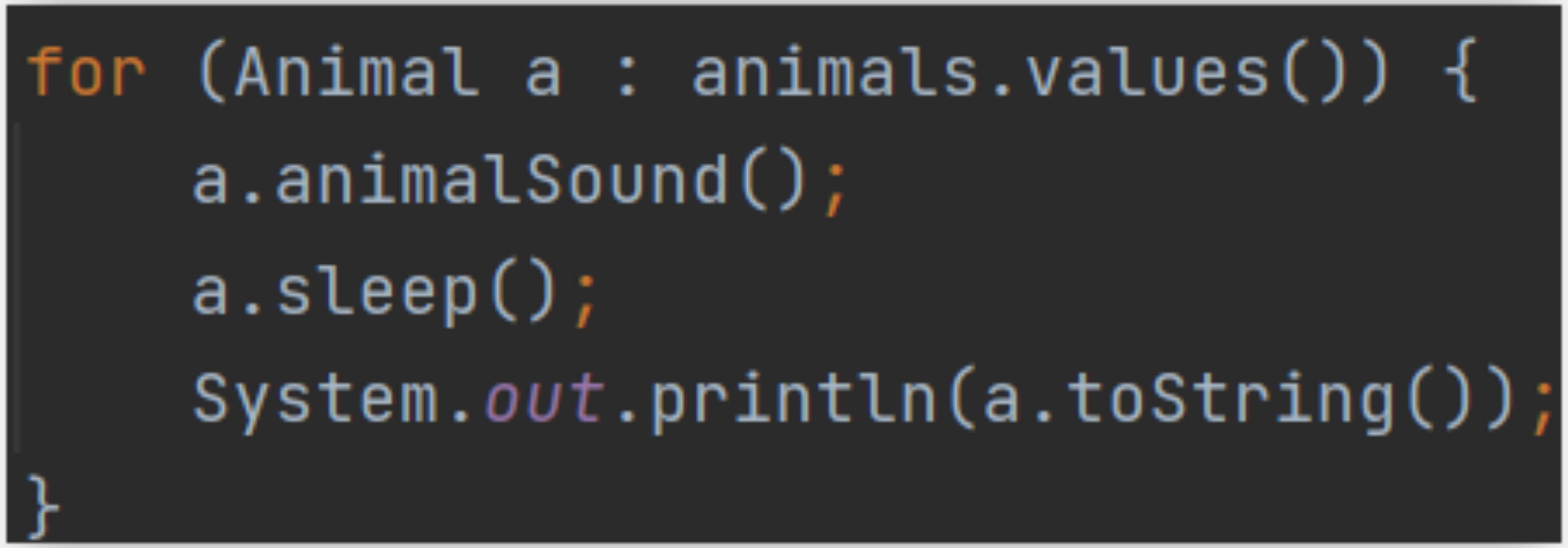
```
capitalCities.remove("England");
```

```
capitalCities.clear();
```

Iterating over objects in a HashMap

- There are lots of possibilities.
- Let’s have a look at one☺

Explanation



```
for (Animal a : animals.values()) {
    a.animalSound();
    a.sleep();
    System.out.println(a.toString());
}
```

Also

- HashMap is nice to use if we want to find an object based on a key.
- If you insert an object with a key that already exists, the value is overwritten.
- We can put objects of defined type and sub-types.
- We have many ways to go through the objects in a HashMap. We've looked at one of them.

Conclusion

- This session's goals:
  - I can use abstract classes, and I understand what that means.
  - I understand what aggregation (and composition) entails.
  - I know how to use a HashMap.

Good luck with the task for this session. Remember you have support from the veiledere.