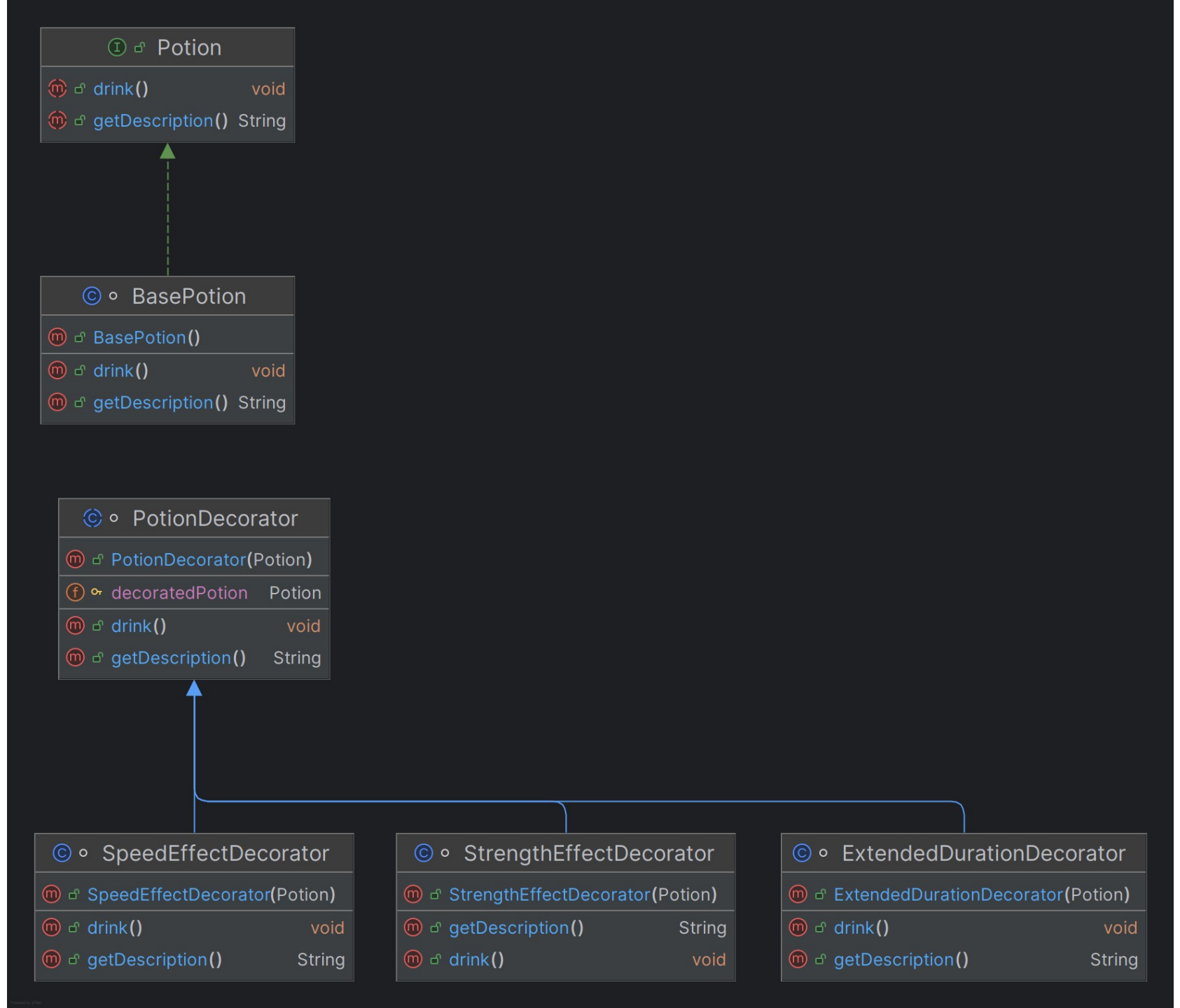
Object-Oriented Programming

4. Lecture
LS 2025/2026
Author: Juraj Petrík

Decorator

- Adding new behavior using wrapping
- Open-Closed Principle


Potion
drink() void
getDescription() String
BasePotion
BasePotion()
drink() void
getDescription() String
PotionDecorator
PotionDecorator(Potion)
decoratedPotion Potion
drink() void
getDescription() String
SpeedEffectDecorator
SpeedEffectDecorator(Potion)
drink() void
getDescription() String
StrengthEffectDecorator
StrengthEffectDecorator(Potion)
getDescription() String
drink() void
ExtendedDurationDecorator
ExtendedDurationDecorator(Potion)
drink() void
getDescription() String

Anti-patterns

- Or how something should not look
- Long Class
- Large Method
- Duplicate code
- Dead Code
- Inappropriate intimacy
- Circular dependency
- …

Refactoring

- Eliminating technical debt
- To keep the code clean
- Clean code =:
  - Lower error rate
  - Easier testability
  - Easier extensibility
  - Easier maintenance

Memory: Stack and Heap

- Memory is freed by the Garbage Collector
  - When no reference points to an object
- Stack
  - Method calls
  - Local variables (a reference to an object is a local variable!)
- Heap
  - All objects

- -Xmx2048m
- -Xms1048m

Fun with ~~flags~~ constructors

- The constructor is called when an object is created (new, but not only new ☺)
- The compiler adds an empty constructor if we have not defined a constructor
- Super() – adds compiler
- This() and Super()
- When inheriting, constructors are executed from "top to bottom" (superclass -> subclass)

Exception handling

- An exception occurs when something undesirable/unexpected/unauthorized
- Try – catch - finally
- Exception – checked (compiler)
- RuntimeException – unchecked, these exceptions should not occur in "production," it is not recommended to catch them
- They are objects, they are polymorphic, from the most specific to the most general

```
try {}
catch (ExceptionA e1) {}
catch (ExceptionB e2) {}
finally {}
```

Exceptions handling

- Throw/Throws
- Handle or declare

Custom Exception

- Extends Exception/RuntimeException

```
public class InvalidFoodException extends Exception { public
   InvalidFoodException(String errorMessage) {
      super(errorMessage);
   }
}
```

```
public Food(String name, int amount, int type, int heal, int feed)
throws InvalidFoodException {
   super(name, amount, type);
   if (amount <= 0 || type <= 0 || heal <= 0 || feed <= 0) {
      throw new InvalidFoodException("Can't construct Food,
can't have a negative amount, type, heal, feed");
   }
   this.setHeal(heal);
   this.setFeed(feed);
}
```

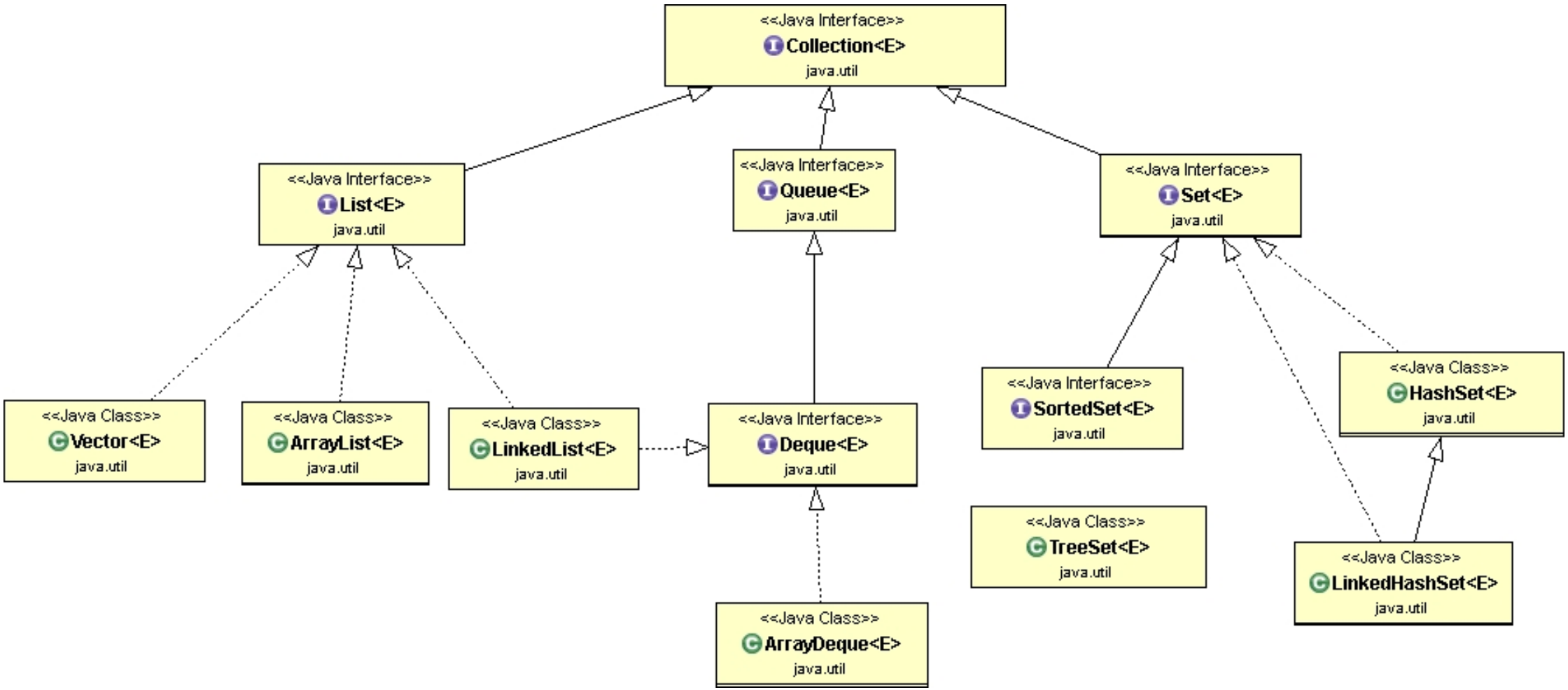
Collections and generics

- List<String> names = new ArrayList<>();
- Collections.sort() -> natural ordering
- Let's sort Steve's

Generics

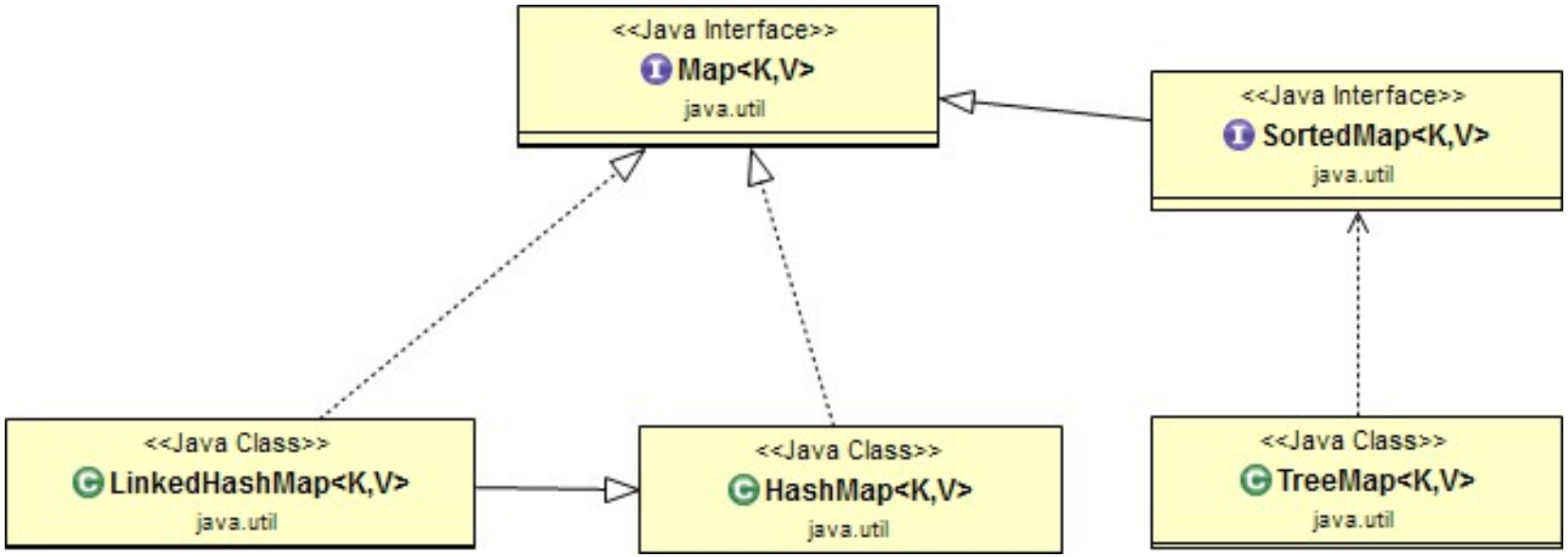
- Parameterized types <Type>
- Generic classes (e.g., ArrayList):
  - Class declaration
  - Declaration of addition methods
  - E is the "type" we want to work with in the class, the compiler will replace
  - E (T, R) is a naming convention
- Generic methods
  - public <T extends SomeClass> void takeThing(ArrayList<T> list)
  - != public void takeThing(ArrayList<SomeClass > list)
- Extends = extends OR implements

List, Queue, Set



Map

- Still considered to belong to Collections



Lambda expressions and double colon operator

- steveArrayList.sort((one, two) -> one.getName().compareTo(two. getName()));
- steveArrayList.sort(Comparator.comparing(Character::getName));

Testing

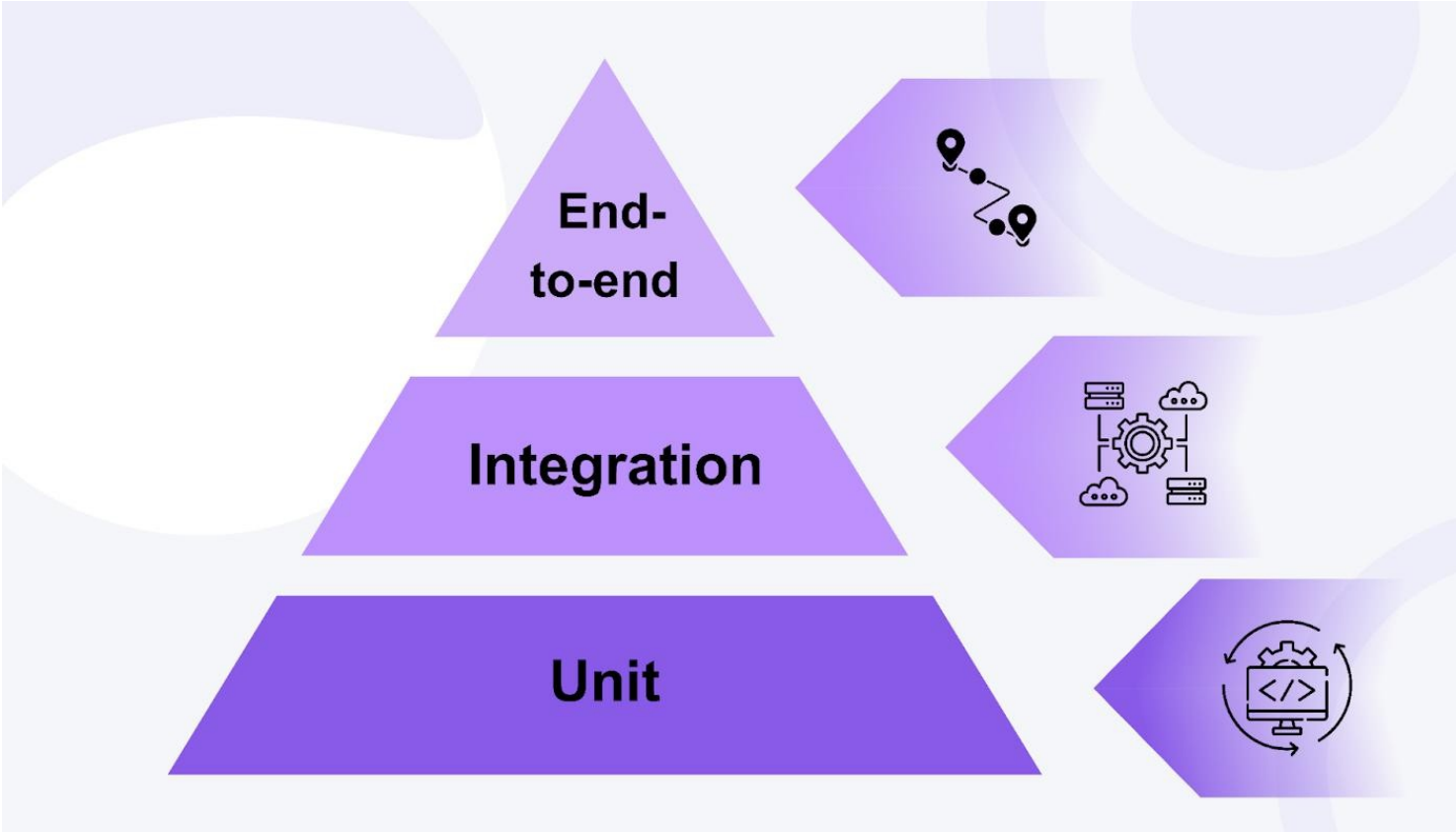
- What types of testing do you know?

Testing

- What kind will we be dealing with here?

Testing

- Unit
- Integration
- Functional
- Acceptance
- Performance
- E2E
- Security



Unit testing (JUnit)

- We test units (the smallest functional units)
- An integral part of development - written by the programmer who implements the given functionality
- The code must be written in such a way that it can be tested
- E.g. with TDD (test-driven development), I write the tests first and then implement them
- We test logic, edge cases, exception handling, etc.
- Assert once

Unit testing

- https://junit.org/junit5/docs/current/user-guide/
- https://github.com/DiUS/java-faker
- https://site.mockito.org/
- https://github.com/Jur1cek/jUnit_examples/

Quiz time