Using JPA

JPA - Java Persistence API

- Standard Java API
- Used for relational databases (that uses SQL)
- ORM - Object Relational Mapping

Object Relational Mapping

- Automatically maps objects to tables in relational databases
- Developers don’t need to write SQL when working with JPA
- Annotations help JPA map objects to tables in relational databases

Spring Data

- Part of the Spring Framework
- Integrates with many kinds of databases (SQL and noSQL)
- Spring Data is a consistent programming model for databases
- Spring Data JPA automates work when using JPA

”Any sufficiently advanced technology is indistinguishable from magic”

Arthur C. Clarke

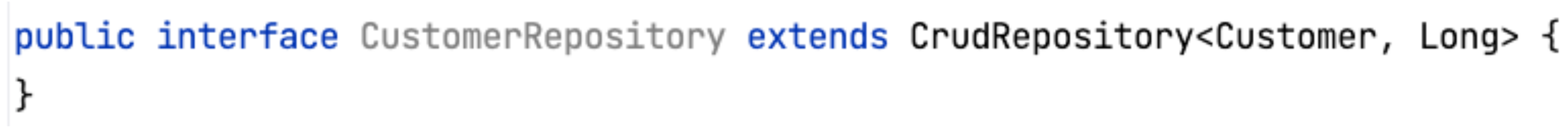
Spring Data JPA

- With Spring Data JPA we don’t need to write SQL
- We don’t even write methods or classes!
- We just create an interface and let Spring Data JPA automatically create an implementation of that interface!

Creating CRUD repositories

- With Spring Data JPA we create an interface for our repository and then Spring Data JPA will create the class that implements it with all the functionality and use an object of that class as a repository

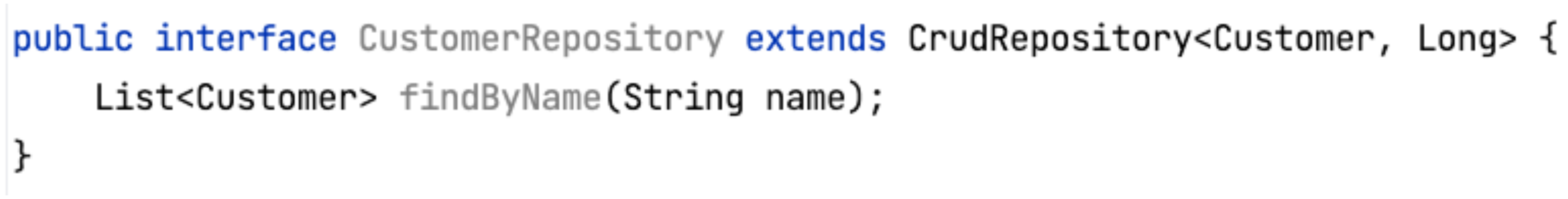
- The interface should extend CrudRepository to inherit many CRUD methods like findAll, findById, deleteById, save and others:



```
public interface CustomerRepository extends CrudRepository<Customer, Long> {
}
```

Creating CRUD repositories

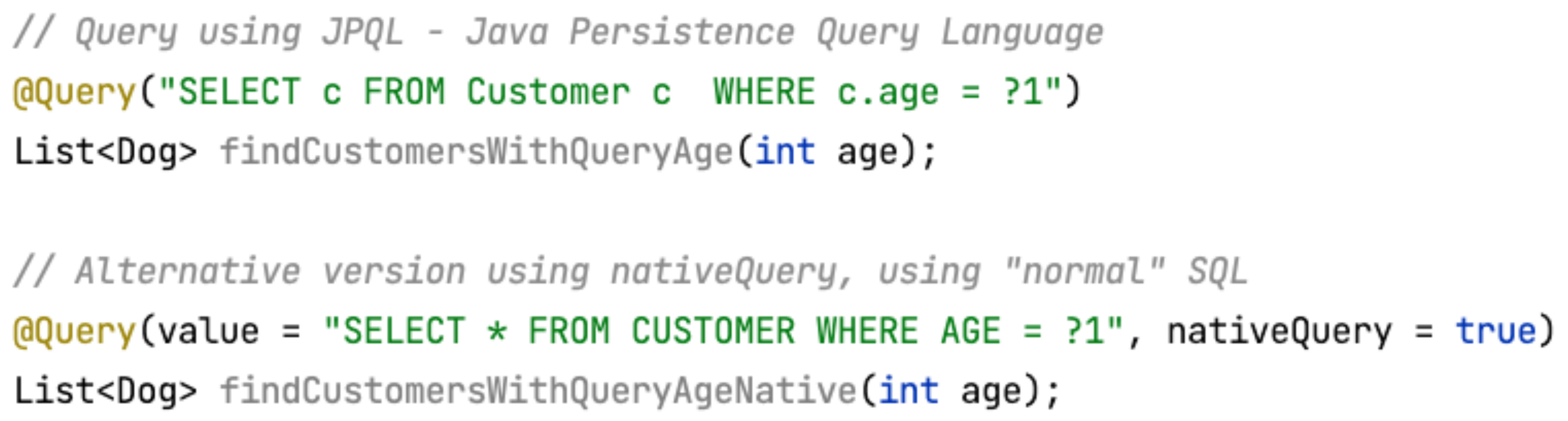
- If you need a specific method that isn't inherited from CrudRepository it is often enough to just add the method signature in the interface. If you give it a name that Spring Data understands it will create the implementation and it will work automatically:



```java
public interface CustomerRepository extends CrudRepository<Customer, Long> {
    List<Customer> findByName(String name);
}
```

Creating CRUD repositories

- You can also specify the implementation of the method by using a Query annotation, either in JPQL or in native SQL:



```java
// Query using JPQL - Java Persistence Query Language
@Query("SELECT c FROM Customer c  WHERE c.age = ?1")
List<Dog> findCustomersWithQueryAge(int age);

// Alternative version using nativeQuery, using "normal" SQL
@Query(value = "SELECT * FROM CUSTOMER WHERE AGE = ?1", nativeQuery = true)
List<Dog> findCustomersWithQueryAgeNative(int age);
```

Demo 1 - Using JPA

- JPA Entity with annotations
- Spring Data Repository (the interface)

Exercise 1 - Using JPA

- Download the UsingJPAStarter project and open it in IntelliJ IDEA
- Change the Dog class into a JPA entity (with annotations)
- Create the Spring Data Repository interface
- Then try out the tests in the test class (uncomment part 1) - all should work!
- Then uncomment part 2 and create the methods in the interface - the tests should all work just by adding the methods in the interface
- If you have time - look at the tests under part 3 and think about what they do, the solutions will be demonstrated in the solutions project