Implementing Declarative Transaction Management

Kesha Williams
SOFTWARE ENGINEERING MANAGER & PROFESSOR

@KeshaWillz www.kesha.tech

Overview

Annotations

- @EnableTransactionManagement
- @Transactional

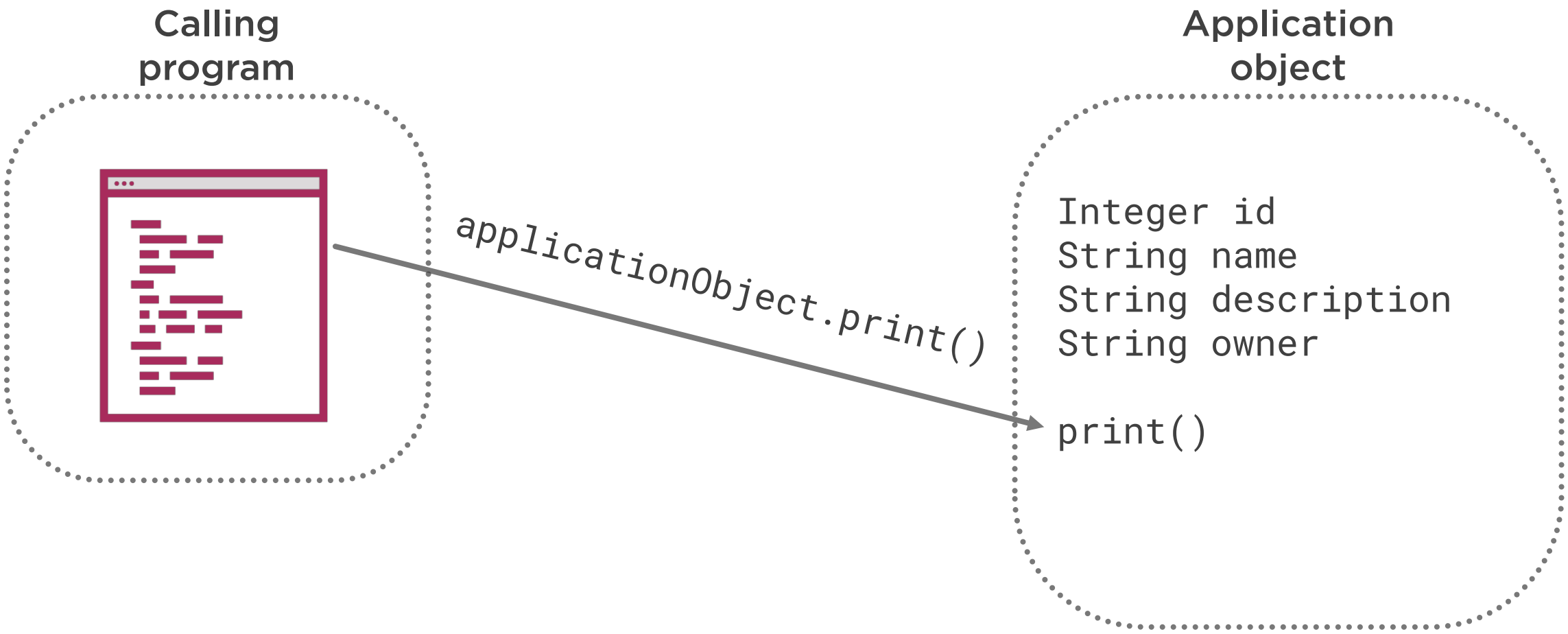
Declarative transaction management

Aspect-oriented programming

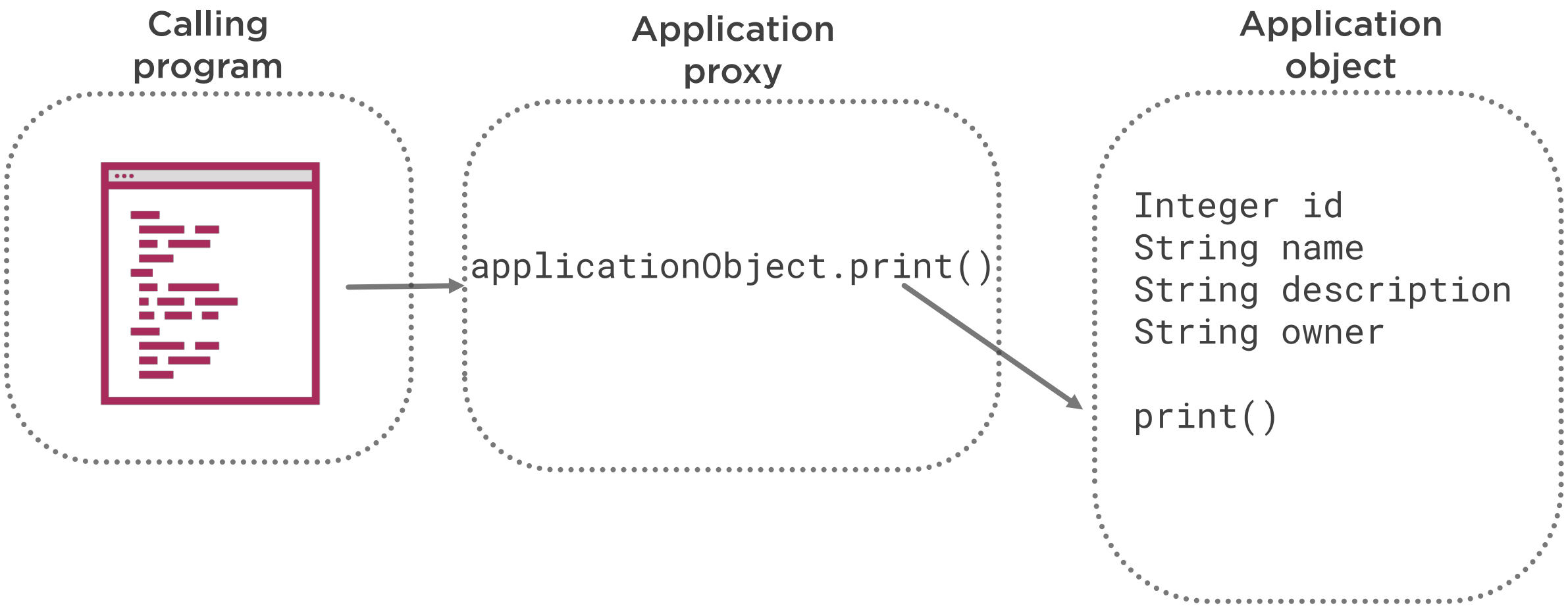
Rollback of transactions

Aspect Oriented Programming (AOP)

Without Proxy



With Proxy


Calling program
Application proxy
applicationObject.print()
Application object
Integer id
String name
String description
String owner
print()

Proxy

Transaction interceptor

Intercepts method calls

Platform transaction manager

Handles transactions

The default advice mode for processing @Transactional annotations is proxy.

@Transactional Overview

Four Components

Persistence context proxy

Entity manager proxy

Transaction aspect

Transaction manager

Entity Manager Proxy

- Entities live in a database
- Manages entities
- Defines methods used
- Persistence context

Persistence Context Proxy

Database transaction

Persistence context

- Set of managed entity instances
- Exist in a data store
- Defines scope for entity instances

Transaction Aspect

Manages transactions

Transaction interceptor

Interceptor called before and after

Two main responsibilities

- Determines if new transaction needed
- Determines to commit, rollback, or leave running

Transaction Manager

Spring's platform transaction manager

JPA transaction manager

Responsible for

- Begin
- Commit
- Rollback

@Transactional Annotation

@Transactional

```
@Service

@Transactional

public class ReleaseService implements IReleaseService{
@Autowired

private ITicketDAO ticketDAO;

@Autowired

private IReleaseDAO releaseDAO;

@Override

public void addRelease (Release release) {
    releaseDAO.addRelease(release);

}
```

Spring recommends that you only annotate concrete classes.

Configuration

XML based

Uses XML files outside of the code to configure

Annotation based

Uses Java annotations in the code to configure

Transaction Advice

```
@Transactional(propagation=…)
```

```
@Transactional ( propagation =
Propagation.REQUIRED )
```

- Code will always run in a transaction

```
@Transactional ( propagation =
Propagation.REQUIRES_NEW )
```

- Code will always run in a new transaction

```
@Transactional ( propagation =
Propagation.NEVER )
```

- Method shouldn’t be run within a transaction

```
@Transactional(isolation=…)
```

```
@Transactional(isolation =
Isolation.READ_UNCOMMITTED)
```

- Allows dirty reads

```
@Transactional(isolation =
Isolation.READ_COMMITTED)
```

- Does not allow dirty reads

```
@Transactional(isolation =
Isolation.REPEATABLE_READ)
```

- Result always the same if row read twice

```
@Transactional(isolation =
Isolation.SERIALIZABLE)
```

- Performs all transactions in a sequence

```
@Transactional(timeout=…)
```

```
@Transactional(timeout=5)
```

- Timeout for the operation wrapped by the transaction

```
@Transactional(timeout=50)
```

```
@Transactional(timeout=30)
```

```
@Transactional(readOnly=…)

@Transactional(read-only = true)
```

- Transactions don’t write back to database
- Optimizes data access
- Provides a hint to the persistence provider
- Only relevant inside a transaction

```
@Transactional(rollbackFor=…,
noRollbackFor=…)
```

```
@Transactional(rollbackFor =
Exception.class)
```

- Define specific exception class that should trigger a rollback

```
@Transactional(rollbackForClassNa
me={"Exception"})
```

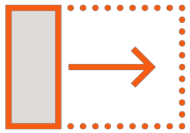
- Define specific class name that should trigger a rollback

```
@Transactional(noRollbackFor =
NoSuchElementException.class)
```

- Specific class name where rollbacks should not occur

Rollback a Declarative Transaction

Rollback



Throw an exception from code

Spring catches any unhandled exception

Spring determines to mark the transaction for rollback

Demo

Enhance Trackzilla

- Declarative transaction management
- Rollback transactions

Summary

Declarative transaction infrastructure

Transaction management support

@Transactional

- readOnly
- noRollback