Good development principles

1. Dependency inversion. Using DAO interface instead of concrete implementation.

public class UserServiceImpl implements UserService {

private UserDao UserDao;

// methods

}

1. Single responsibility. Using separate Service for manipulation each bean.

public class EventServiceImpl implements EventService {

@Override

public Event createEvent(Event event) {...}

@Override

public Event updateEvent(Event event) {...}

@Override

public boolean deleteEvent(long eventId) {...}

}

1. Dry principle. Using aspect to avoid code duplication.

@Aspect

public class MethodLogger {

private static Logger LOG = Logger.getLogger(MethodLogger.class);

@Around("execution(\* com.epam.hw1.dao.\*.\*(..))")

public Object logAroundMethod(ProceedingJoinPoint point){

// logging

}

}

Bad practice

1. Single responsibility principle violation. Using String for address instead of Address abstraction.

public class User {

public void setId(long id){...}

public void setName(String name){...}

public void setAddress(String address){...}

}

We can create class Address with certain properties:

public class User {

public void setId(long id){...}

public void setName(String name){...}

public void setAddress(Address address){...}

}

public class Address {

private String street;

private String flat;

}

2. Dry principle violation. Manual transaction manipulation for each query to DB.

DefaultTransactionDefinition definition = new DefaultTransactionDefinition();

definition.setPropagationBehavior(TransactionDefinition.PROPAGATION\_REQUIRES);

TransactionStatus status = transactionManager.getTransaction(definition);

try {

namedParamJdbcTemplate.batchUpdate(SQL\_INSERT, params);

transactionManager.commit(status);

} catch (DataAccessException e) {

transactionManager.rollback(status);

}

We can use @Transactional instead to reduce code duplication.

@Transactional

public boolean cancelTicket(long ticketId) { ... }

1. Liskov substitution principle violation. JsonParser should implement interface but not throw exception instead of it.

public interface Parser {

Object parseFromString (String string);

String parseToString (String content);

}

public class JsonParser implements Parser {

public Object parseFromString (String string) {...}

public String parseToString (String content) { throw new UncupportedOperationException();}

}

Jsonparser should implement interface that’s completely fit it.

public interface Parser {

Object parse (String string);

}

public class JsonParser implements Parser {

public Object parseFromString (String string) {...}

}