prepare for…

Functional-Reactive

…with Core Java 8/9

@SvenRuppert

has been coding java since 1996

Head of R&D

reply Group

Germany - Munich

@SvenRuppert

has been coding java since 1996

@SvenRuppert

has been coding java since 1996

Projects in the field of:

- Automobile-industry
- Energy
- Finance / Leasing
- Space- Satellit-
- Government / UN / World-bank

Where?

- Europe
- Asia - from India up to Malaysia

Save harbor statement

Save harbor statement

The following is intended for information purposes only. I can not be held responsible for the overuse of effects and animations in this presentation. If any person in this room has a medical condition that is triggered by fast moving objects on the screen and/or explosions, he/she should probably better leave now...

(I got carried away by the topic.)

Small AddOns

based on Java8

@SvenRuppert

Java8 - Optional<T>

Java8 - Optional<T>

@SvenRuppert

```
final boolean present = stringOptional.isPresent();
stringOptional.ifPresent(s1 -> System.out.println("stored = " + s1));
```

Java8 - Optional<T>

@SvenRuppert

```
final boolean present = stringOptional.isPresent();
stringOptional.ifPresent(s1 -> System.out.println("stored = " + s1));
```

```
final String orElseThrow = stringOptional.orElseThrow(() -> new RuntimeException("to bad.."));
```

Java8 - Optional<T>

@SvenRuppert

```
final boolean present = stringOptional.isPresent();
stringOptional.ifPresent(s1 -> System.out.println("stored = " + s1));
```

```
final String orElseThrow = stringOptional.orElseThrow(() -> new RuntimeException("to bad.."));
```

```
final Optional<String> s1 = stringOptional.filter(Objects::isNull); // only with value if Predicate true
```

Java8 - Optional<T>

@SvenRuppert

```
final boolean present = stringOptional.isPresent();
stringOptional.ifPresent(s1 -> System.out.println("stored = " + s1));
```

```
final String orElseThrow = stringOptional.orElseThrow(() -> new RuntimeException("to bad.."));
```

```
final Optional<String> s1 = stringOptional.filter(Objects::isNull); // only with value if Predicate true
```

```
final Optional<Integer> flatMapOptional = stringOptional.flatMap(new Function<String, Optional<Integer>>() {
  @Override
  public Optional<Integer> apply(final String s) {
    if (Objects.isNull(s)) return Optional.empty();
    return Optional.of(s.length());
  }
});
```

Java8 - Optional<T>

@SvenRuppert

```
final boolean present = stringOptional.isPresent();
stringOptional.ifPresent(s1 -> System.out.println("stored = " + s1));
```

```
final String orElseThrow = stringOptional.orElseThrow(() -> new RuntimeException("to bad.."));
```

```
final Optional<String> s1 = stringOptional.filter(Objects::isNull); // only with value if Predicate true
```

```
final Optional<Integer> flatMapOptional = stringOptional.flatMap(new Function<String, Optional<Integer>>() {
  @Override
  public Optional<Integer> apply(final String s) {
    if (Objects.isNull(s)) return Optional.empty();
    return Optional.of(s.length());
  }
});
```

Optional - [Int, Double, Long]

Java8 - Functional Interfaces

Java8 - Functional Interfaces

@SvenRuppert

```
@FunctionalInterface
public interface DemoInterface {
    public void doSomething();
}
```

Java8 - Functional Interfaces

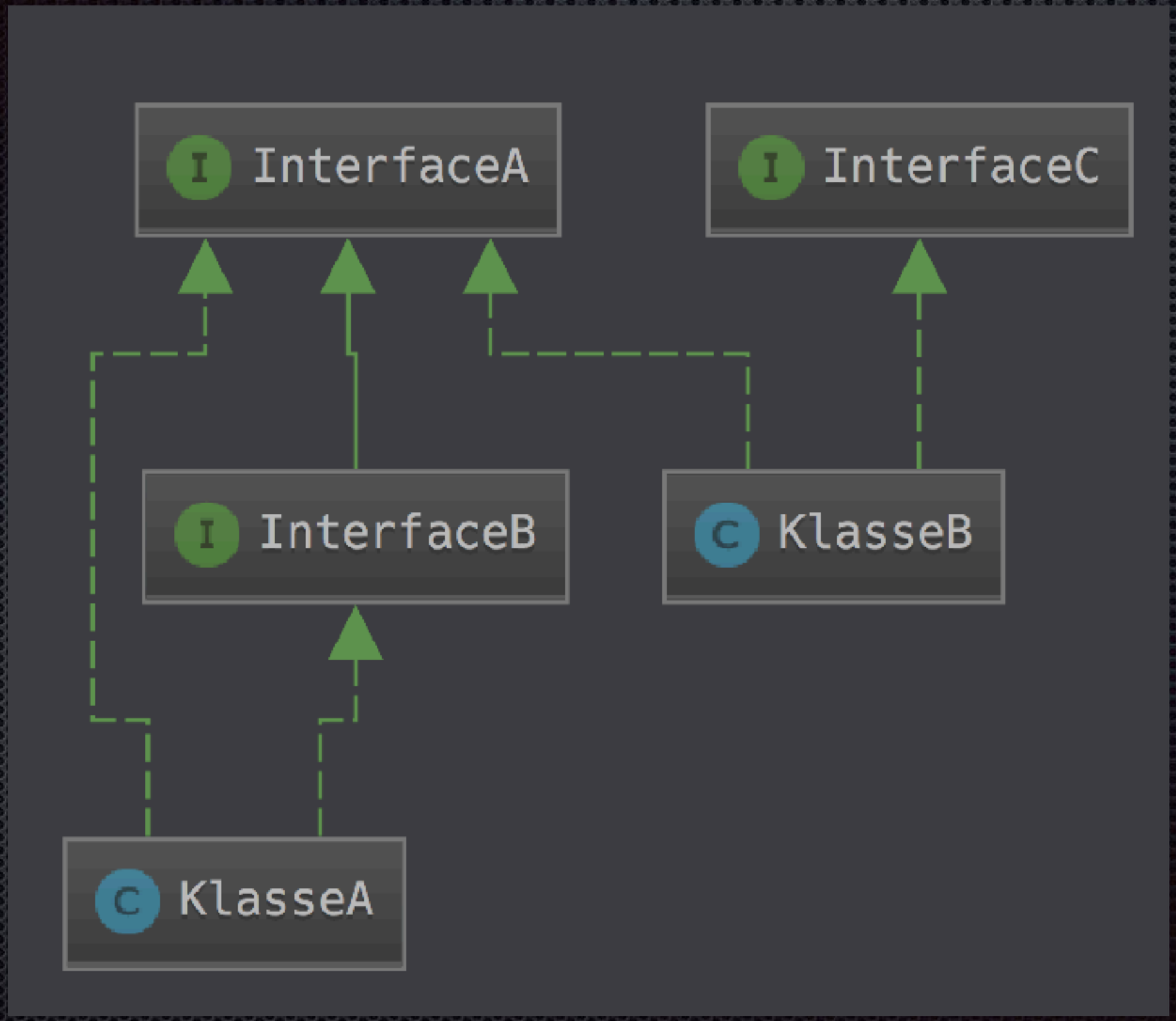
@SvenRuppert

```java
@FunctionalInterface
public interface DemoInterface {
    public void doSomething();
}
```

```java
@FunctionalInterface
public interface InterfaceA {
    public String doMoreB();

    public default String doMoreA(){
        return "aaeettsch";
    }
}
```

Java8 - Functional Interfaces

@SvenRuppert



Java8 - Functional Interfaces

Java8 - Functional Interfaces

@SvenRuppert

```
public interface InterfaceA {

    public static void doMore() { System.out.println(" doMore "); }

    public default void doSomething() { System.out.println(" InterfaceA -> doSomething"); }
}
```

Java8 - Functional Interfaces

@SvenRuppert

```
public interface InterfaceA {

    public static void doMore() { System.out.println(" doMore "); }

    public default void doSomething() { System.out.println(" InterfaceA -> doSomething"); }
}
```

```
public interface InterfaceB extends InterfaceA {
    public default void doSomething() { System.out.println(" InterfaceB -> doSomething"); }
}
```

Java8 - Functional Interfaces

@SvenRuppert

```
public interface InterfaceA {

    public static void doMore() { System.out.println(" doMore "); }

    public default void doSomething() { System.out.println(" InterfaceA -> doSomething"); }
}
```

```
public interface InterfaceB extends InterfaceA {
    public default void doSomething() { System.out.println(" InterfaceB -> doSomething"); }
}
```

```
public interface InterfaceC {
    public  default void doSomething() { System.out.println(" InterfaceC -> doSomething"); }
}
```

Java8 - Functional Interfaces

@SvenRuppert

```
public interface InterfaceA {

    public static void doMore() { System.out.println(" doMore "); }

    public default void doSomething() { System.out.println(" InterfaceA -> doSomething"); }
}
```

```
public interface InterfaceB extends InterfaceA {
    public default void doSomething() { System.out.println(" InterfaceB -> doSomething"); }
}
```

```
public interface InterfaceC {
    public  default void doSomething() { System.out.println(" InterfaceC -> doSomething"); }
}
```

```
public class KlasseA implements InterfaceA, InterfaceB {

}
```

Java8 - Functional Interfaces

@SvenRuppert

```java
public interface InterfaceA {

    public static void doMore() { System.out.println(" doMore "); }

    public default void doSomething() { System.out.println(" InterfaceA -> doSomething"); }
}
```

```java
public interface InterfaceB extends InterfaceA {
    public default void doSomething() { System.out.println(" InterfaceB -> doSomething"); }
}
```

```java
public interface InterfaceC {
    public  default void doSomething() { System.out.println(" InterfaceC -> doSomething"); }
}
```

```java
public class KlasseA implements InterfaceA, InterfaceB {

}
```

```java
public class KlasseB implements InterfaceA, InterfaceC {

    @Override
    public void doSomething() {
        InterfaceA a = new InterfaceA() {};
        a.doSomething();
    }
}
```

Java8 - Functional Interfaces

@SvenRuppert

```java
public interface InterfaceA {

    public static void doMore() { System.out.println(" doMore "); }

    public default void doSomething() { System.out.println(" InterfaceA -> doSomething"); }
}
```

```java
public interface InterfaceB extends InterfaceA {
    public default void doSomething() { System.out.println(" InterfaceB -> doSomething"); }
}
```

```java
public interface InterfaceC {
    public  default void doSomething() { System.out.println(" InterfaceC -> doSomething"); }
}
```

```java
public class KlasseA implements InterfaceA, InterfaceB {

}
```

```java
public class KlasseB implements InterfaceA, InterfaceC {

    @Override
    public void doSomething() {
        InterfaceA a = new InterfaceA() {};
        a.doSomething();
    }
}
```

```java
KlasseA a = new KlasseA();
a.doSomething();
//KlasseA.doMore();

final KlasseB b = new KlasseB();
b.doSomething();

InterfaceA.doMore();
```

Java8 - Functional Interfaces

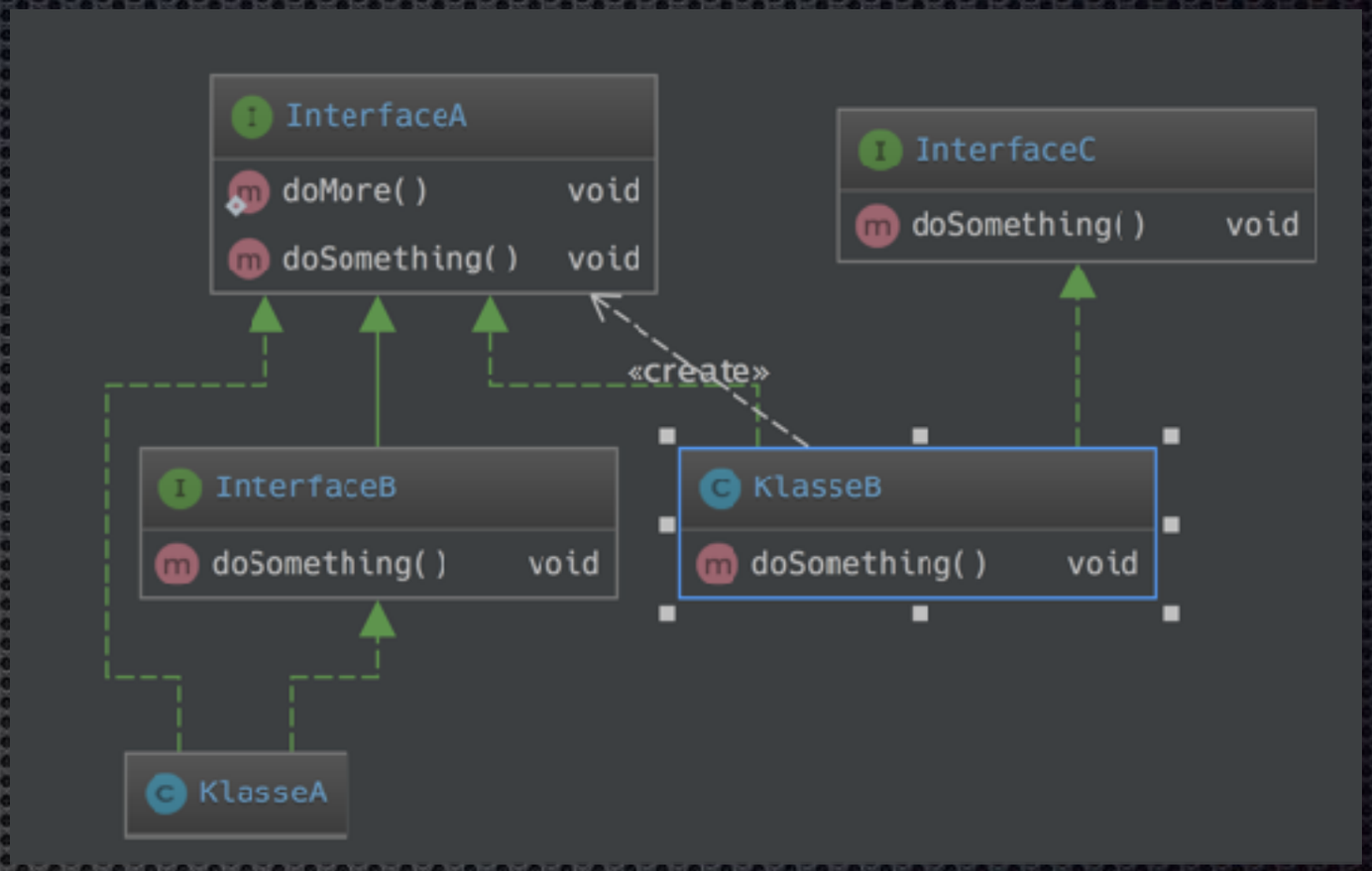
@SvenRuppert

```
KlasseA a = new KlasseA();
a.doSomething();
//KlasseA.doMore();

final KlasseB b = new KlasseB();
b.doSomething();

InterfaceA.doMore();
```

Java8 - Functional Interfaces

@SvenRuppert

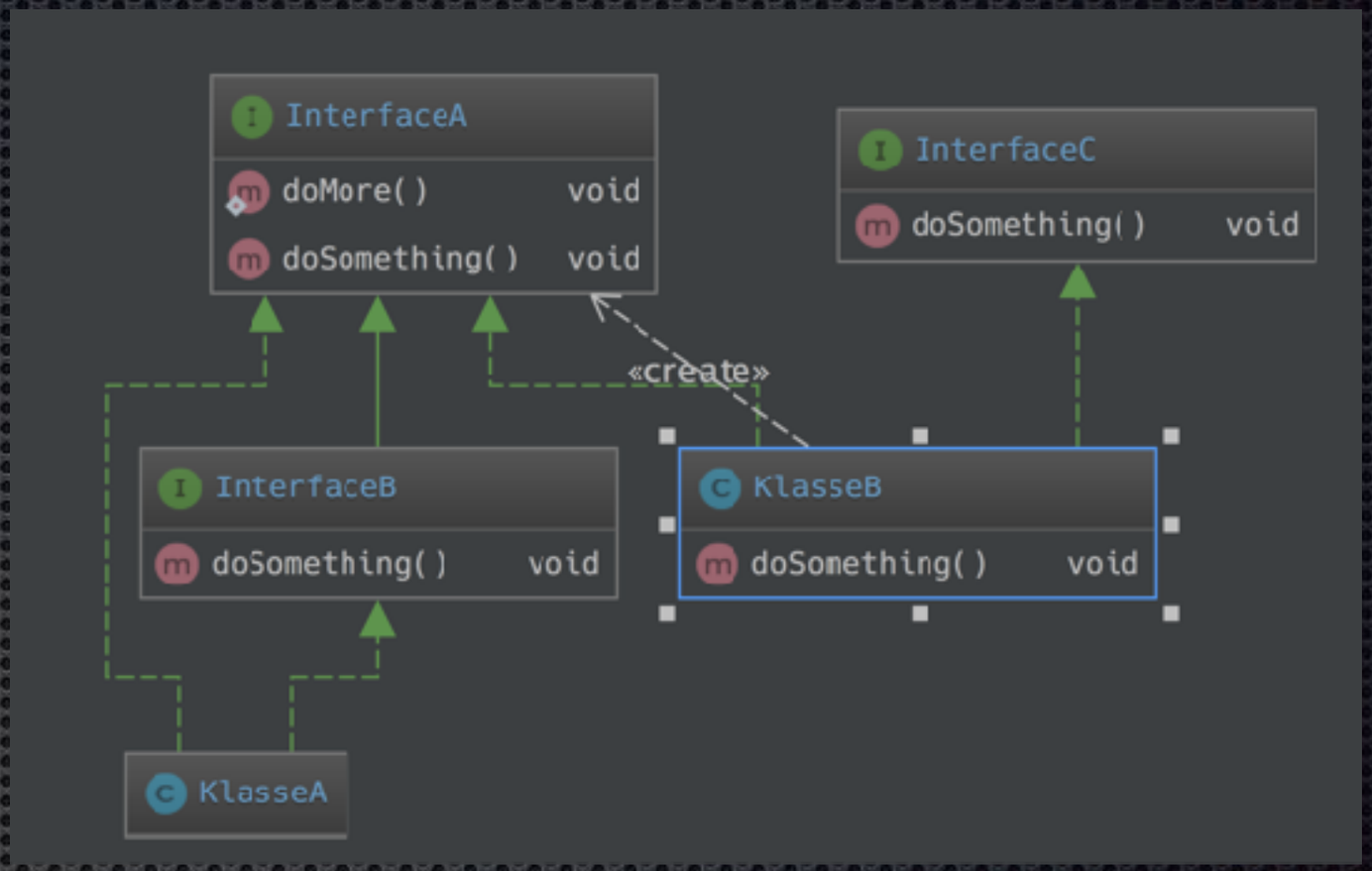
```
KlasseA a = new KlasseA();
a.doSomething();
//KlasseA.doMore();

final KlasseB b = new KlasseB();
b.doSomething();

InterfaceA.doMore();
```



Java8 - Functional Interfaces

@SvenRuppert

```
KlasseA a = new KlasseA();
a.doSomething();
//KlasseA.doMore();

final KlasseB b = new KlasseB();
b.doSomething();

InterfaceA.doMore();
```




```
InterfaceB -> doSomething
InterfaceA -> doSomething
doMore
```

Java9 - Functional Interfaces

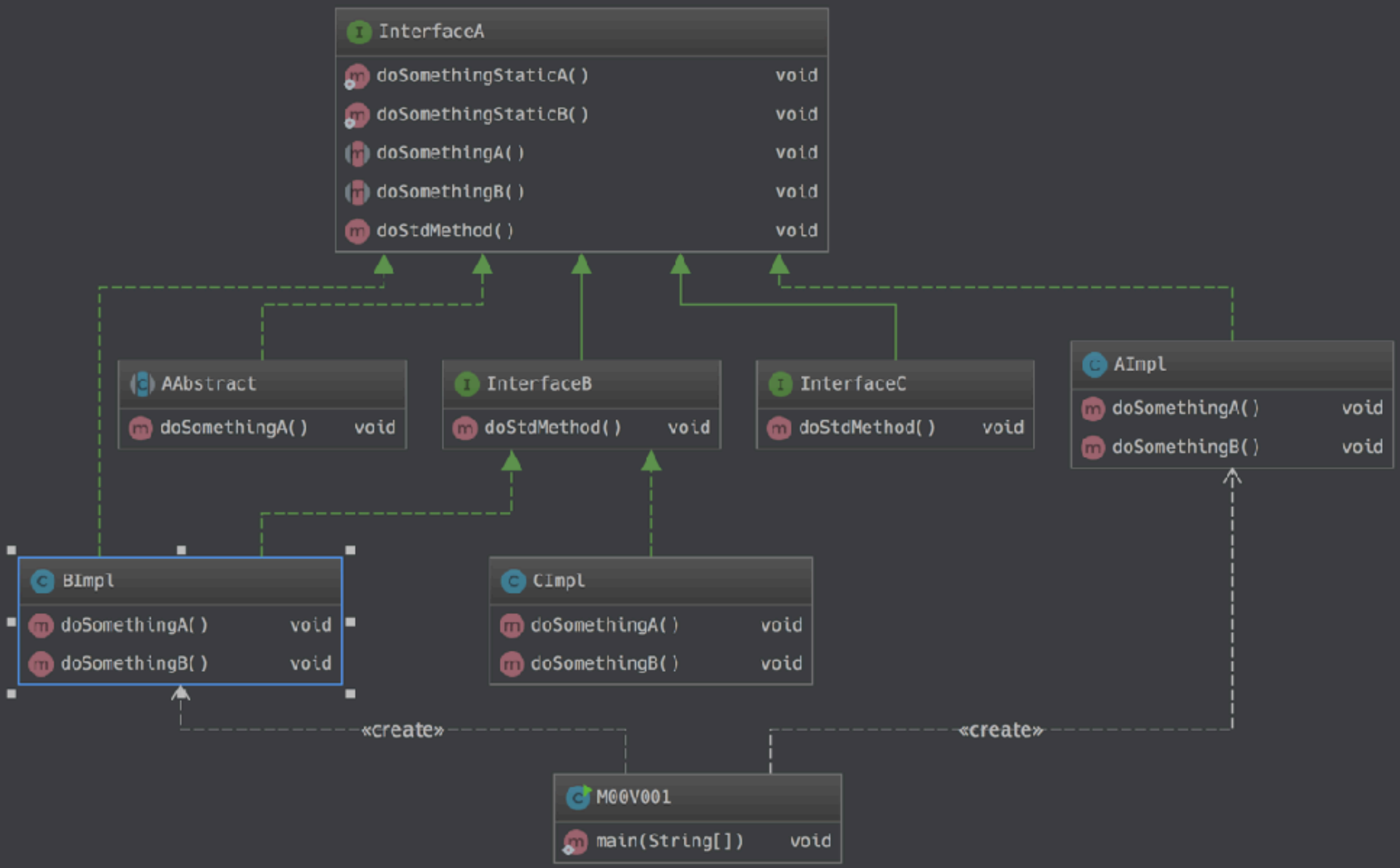
@SvenRuppert

Java9 - JEP213

```
@FunctionalInterface
public interface Service {

  default void workOnDefault() {
    workA();
    workB();
    workC();
    //somethingHidden();
  }

  void workA();

  private void workB() { System.out.println("workB"); }

  private static void workC() { System.out.println("workC"); }

  //private void somethingHidden(); // declaration only not possible
}
```

Java8 - Functional Interfaces - Try it

@SvenRuppert



Java8 - Functional Interfaces - Example

@SvenRuppert

typical legacy implementation

```
public void writeUser(final User user){
  final int update = update(user, connectionPool());
  // check result
}
```

```
private int update(User user, final JDBCConnectionPool connectionPool) {
  final DataSource dataSource = connectionPool.getDataSource();
  try {
    final Connection connection = dataSource.getConnection();
    final int count;
    try (final Statement statement = connection.createStatement()) {
      final String sql = createUpdteSQL(user);
      count = statement.executeUpdate(sql);
      statement.close();
    }
    dataSource.evictConnection(connection);
    return count;
  } catch (final SQLException e) {
    e.printStackTrace();
  }
  return -1;
}
```

Java8 - Functional Interfaces - Example

@SvenRuppert

typical legacy implementation

```
private String createUpdteSQL(final User user) {
  return ""; // vendor SQL , MySQL/Oracle...
}


public static class User { }

public abstract JDBCConnectionPool connectionPool();

public interface JDBCConnectionPool {...}

public interface DataSource {...}
```

Java8 - Functional Interfaces - Example

Java8 - Functional Interfaces - Example

@SvenRuppert

```
@FunctionalInterface
public interface BasicOperation {
  @NotNull
  String createSQL();
}
```

Java8 - Functional Interfaces - Example

@SvenRuppert

```java
@FunctionalInterface
public interface BasicOperation {
  @NotNull
  String createSQL();
}
```

```java
@FunctionalInterface
public interface Update extends BasicOperation {

  default int update(final JDBCConnectionPool connectionPool) {
    final HikariDataSource dataSource = connectionPool.getDataSource();
    try {
      final Connection connection = dataSource.getConnection();
      final int count;
      try (final Statement statement = connection.createStatement()) {
        final String sql = createSQL();
        count = statement.executeUpdate(sql);
        statement.close();
      }
      dataSource.evictConnection(connection);
      return count;
    } catch (final SQLException e) {
      e.printStackTrace();
    }
    return -1;
  }

}
```

Java8 - Functional Interfaces - Example

@SvenRuppert

```java
@FunctionalInterface
public interface BasicOperation {
  @NotNull
  String createSQL();
}
```

```java
@FunctionalInterface
public interface Update extends BasicOperation {

  default int update(final JDBCConnectionPool connectionPool) {
    final HikariDataSource dataSource = connectionPool.getDataSource();
    try {
      final Connection connection = dataSource.getConnection();
      final int count;
      try (final Statement statement = connection.createStatement()) {
        final String sql = createSQL();
        count = statement.executeUpdate(sql);
        statement.close();
      }
      dataSource.evictConnection(connection);
      return count;
    } catch (final SQLException e) {
      e.printStackTrace();
    }
    return -1;
  }

}
```

Java8 - Functional Interfaces - Example

@SvenRuppert

```java
@FunctionalInterface
public interface BasicOperation {
  @NotNull
  String createSQL();
}
```

```java
@FunctionalInterface
public interface Update extends BasicOperation {

  default int update(final JDBCConnectionPool connectionPool) {
    final HikariDataSource dataSource = connectionPool.getDataSource();
    try {
      final Connection connection = dataSource.getConnection();
      final int count;
      try (final Statement statement = connection.createStatement()) {
        final String sql = createSQL();
        count = statement.executeUpdate(sql);
        statement.close();
      }
      dataSource.evictConnection(connection);
      return count;
    } catch (final SQLException e) {
      e.printStackTrace();
    }
    return -1;
  }

}
```

Java8 - Functional Interfaces - Example

@SvenRuppert

```java
@FunctionalInterface
public interface BasicOperation {
  @NotNull
  String createSQL();
}
```

```java
@FunctionalInterface
public interface Update extends BasicOperation {

  default int update(final JDBCConnectionPool connectionPool) {
    final HikariDataSource dataSource = connectionPool.getDataSource();
    try {
      final Connection connection = dataSource.getConnection();
      final int count;
      try (final Statement statement = connection.createStatement()) {
        final String sql = createSQL();
        count = statement.executeUpdate(sql);
        statement.close();
      }
      dataSource.evictConnection(connection);
      return count;
    } catch (final SQLException e) {
      e.printStackTrace();
    }
    return -1;
  }

}
```

Java8 - Functional Interfaces - Example

Java8 - Functional Interfaces - Example

@SvenRuppert

```
@FunctionalInterface
public interface DAO {
  JDBCConnectionPool connectionPool();
}
```

Java8 - Functional Interfaces - Example

@SvenRuppert

```
@FunctionalInterface
public interface DAO {
  JDBCConnectionPool connectionPool();
}
```

```
public abstract class GenericDAO implements DAO {

  private JDBCConnectionPool connectionPool;

  public DAO workOnPool(final JDBCConnectionPool connectionPool) {
    this.connectionPool = connectionPool;
    return this;
  }

  @Override
  public JDBCConnectionPool connectionPool() { return connectionPool; }
}
```

Java8 - Functional Interfaces - Example

@SvenRuppert

```
@FunctionalInterface
public interface DAO {
  JDBCConnectionPool connectionPool();
}
```

```
public abstract class GenericDAO implements DAO {

  private JDBCConnectionPool connectionPool;

  public DAO workOnPool(final JDBCConnectionPool connectionPool) {
    this.connectionPool = connectionPool;
    return this;
  }

  @Override
  public JDBCConnectionPool connectionPool() { return connectionPool; }
}
```

Java8 - Functional Interfaces - Example

@SvenRuppert

```
@FunctionalInterface
public interface DAO {
  JDBCConnectionPool connectionPool();
}
```

```
public abstract class GenericDAO implements DAO {

  private JDBCConnectionPool connectionPool;

  public DAO workOnPool(final JDBCConnectionPool connectionPool) {
    this.connectionPool = connectionPool;
    return this;
  }

  @Override
  public JDBCConnectionPool connectionPool() { return connectionPool; }
}
```

Java8 - Functional Interfaces - Example

Java8 - Functional Interfaces - Example

@SvenRuppert

```
public interface UserDAO extends DAO {

  default void writeUser(final User user) {
    ((Update) () -> createInsert(user)).update(connectionPool());
  }

  String createInsert(User user);
```

Java8 - Functional Interfaces - Example

@SvenRuppert

```
public interface UserDAO extends DAO {

  default void writeUser(final User user) {
    ((Update) () -> createInsert(user)).update(connectionPool());
  }

  String createInsert(User user);
```

Java8 - Functional Interfaces - Example

Java8 - Functional Interfaces - Example

@SvenRuppert

```
public class UserDAOHSQL extends GenericDAO implements UserDAO {

  public String createInsert(final User user) {
    final String sql = "INSERT INTO CUSTOMER " +
        "( CUSTOMER_ID, FIRSTNAME, LASTNAME, EMAIL ) " +
        " VALUES " +
        "( " +
        user.getCustomerID() + ", " +
        "'" + user.getFirstname() + "', " +
        "'" + user.getLastname() + "', " +
        "'" + user.getEmail() + "' " +
        ")";
    return sql;
  }
```

Java8 - Functional Interfaces - Example

@SvenRuppert

```
public class UserDAOHSQL extends GenericDAO implements UserDAO {

  public String createInsert(final User user) {
    final String sql = "INSERT INTO CUSTOMER " +
        "( CUSTOMER_ID, FIRSTNAME, LASTNAME, EMAIL ) " +
        " VALUES " +
        "( " +
        user.getCustomerID() + ", " +
        "'" + user.getFirstname() + "', " +
        "'" + user.getLastname() + "', " +
        "'" + user.getEmail() + "' " +
        ")";
    return sql;
  }
```

Java8 - Functional Interfaces - Example

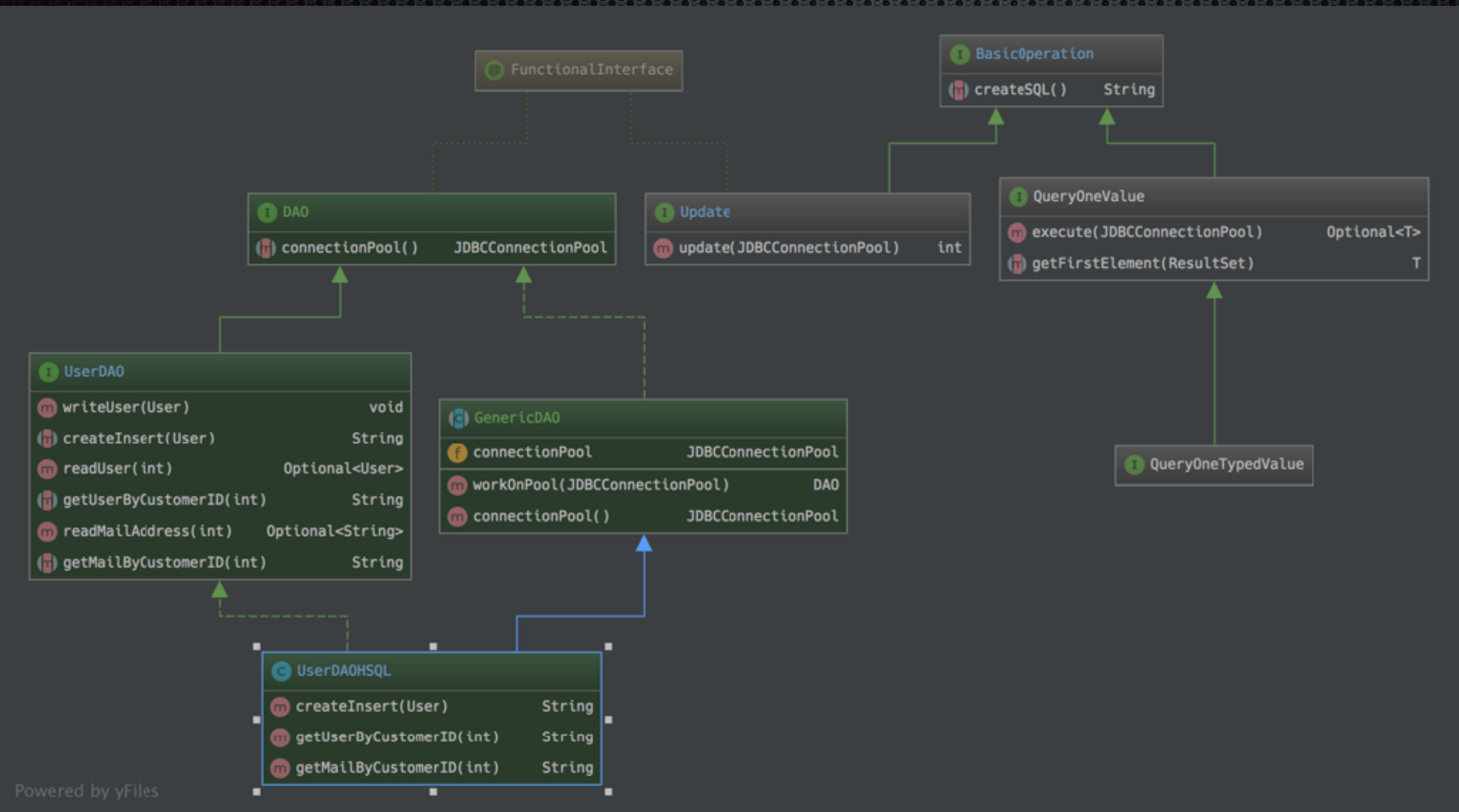
@SvenRuppert

```
public class UserDAOHSQL extends GenericDAO implements UserDAO {

  public String createInsert(final User user) {
    final String sql = "INSERT INTO CUSTOMER " +
        "( CUSTOMER_ID, FIRSTNAME, LASTNAME, EMAIL ) " +
        " VALUES " +
        "( " +
        user.getCustomerID() + ", " +
        "'" + user.getFirstname() + "', " +
        "'" + user.getLastname() + "', " +
        "'" + user.getEmail() + "' " +
        ")";
    return sql;
  }
```

Java8 - Functional Interfaces - Example

Java8 - Functional Interfaces - Example

@SvenRuppert



Java8 - Lambdas

Java8 - Lambdas

@SvenRuppert

```
public static void doMore(DemoInterface d) {
    d.doSomething();
}

public static void printMe(){
    System.out.println("printMe");
}
```

Java8 - Lambdas

@SvenRuppert

```
public static void doMore(DemoInterface d) {
    d.doSomething();
}

public static void printMe(){
    System.out.println("printMe");
}
```

```
doMore(() -> {
            System.out.println("new Date() = " + new Date());
        });
```

Java8 - Lambdas

@SvenRuppert

```
public static void doMore(DemoInterface d) {
    d.doSomething();
}

public static void printMe(){
    System.out.println("printMe");
}
```

```
doMore(() -> {
            System.out.println("new Date() = " + new Date());
        });
```

```
doMore(() -> System.out.println("new Date() = " + new Date()));
```

Java8 - Lambdas

@SvenRuppert

```
public static void doMore(DemoInterface d) {
    d.doSomething();
}

public static void printMe(){
    System.out.println("printMe");
}
```

```
doMore(() -> {
        System.out.println("new Date() = " + new Date());
    });
```

```
doMore(() -> System.out.println("new Date() = " + new Date()));
```

```
doMore(new DemoInterface() {
    @Override
    public void doSomething() { printMe(); }
});
```

Java8 - Lambdas

@SvenRuppert

```
public static void doMore(DemoInterface d) {
    d.doSomething();
}

public static void printMe(){
    System.out.println("printMe");
}
```

Java8 - Lambdas

@SvenRuppert

```
public static void doMore(DemoInterface d) {
    d.doSomething();
}

public static void printMe(){
    System.out.println("printMe");
}
```

```
doMore(new DemoInterface() {
    @Override
    public void doSomething() { printMe(); }
});
```

Java8 - Lambdas

@SvenRuppert

```
public static void doMore(DemoInterface d) {
    d.doSomething();
}

public static void printMe(){
    System.out.println("printMe");
}
```

```
doMore(() -> {
    printMe();
});
```

```
doMore(new DemoInterface() {
    @Override
    public void doSomething() { printMe(); }
});
```

Java8 - Lambdas

@SvenRuppert

```
public static void doMore(DemoInterface d) {
    d.doSomething();
}

public static void printMe(){
    System.out.println("printMe");
}
```

```
doMore(() -> {
    printMe();
});
```

```
doMore(new DemoInterface() {
    @Override
    public void doSomething() { printMe(); }
});
```

```
doMore(() -> printMe());
doMore(Main::printMe);
```

Pattern in Java 8/9

how pattern will change…

@SvenRuppert

Virtual Proxy

Virtual Proxy

@SvenRuppert

```
public class ServiceProxy implements Service {
  private Service service = new ServiceImpl();
  public String work(String txt) {  return service.work(txt); }
}
```

Virtual Proxy

@SvenRuppert

```
public class ServiceProxy implements Service {
  private Service service = new ServiceImpl();
  public String work(String txt) {  return service.work(txt); }
}
```

What could we change now ?

Virtual Proxy

@SvenRuppert

```
public class ServiceProxy implements Service {
  private Service service = new ServiceImpl();
  public String work(String txt) {  return service.work(txt); }
}
```

What could we change now ?

Virtual Proxy:

create the Delegator later

Virtual Proxy

@SvenRuppert

```
public class ServiceProxy implements Service {
  private Service service = new ServiceImpl();
  public String work(String txt) {  return service.work(txt); }
}
```

What could we change now ?

Virtual Proxy:

create the Delegator later

```
public class VirtualService implements Service {
  private Service service = null;
  public String work(String txt) {
    if(service == null) { service = new ServiceImpl(); }
    return service.work(txt);
  }
}
```

Virtual Proxy

@SvenRuppert

```
public class ServiceProxy implements Service {
  private Service service = new ServiceImpl();
  public String work(String txt) {  return service.work(txt); }
}
```

What could we change now ?

Virtual Proxy:

create the Delegator later

```
public class VirtualService implements Service {
  private Service service = null;
  public String work(String txt) {
    if(service == null) { service = new ServiceImpl(); }
    return service.work(txt);
  }
}
```

Virtual Proxy

@SvenRuppert

```
public class ServiceProxy implements Service {
  private Service service = new ServiceImpl();
  public String work(String txt) {  return service.work(txt); }
}
```

What could we change now ?

Virtual Proxy:

create the Delegator later

```
public class VirtualService implements Service {
  private Service service = null;
  public String work(String txt) {
    if(service == null) { service = new ServiceImpl(); }
    return service.work(txt);
  }
}
```

Virtual Proxy

@SvenRuppert

```
public class VirtualService implements Service {
  private Service service = null;
  public String work(String txt) {
    if(service == null) { service = new ServiceImpl(); }

    return service.work(txt);
  }
}
```

Virtual Proxy

@SvenRuppert

```
public class VirtualService implements Service {
  private Service service = null;
  public String work(String txt) {
    if(service == null) { service = new ServiceImpl(); }

    return service.work(txt);
  }
}
```

Virtual Proxy

@SvenRuppert

```
public class VirtualService implements Service {
  private Service service = null;
  public String work(String txt) {
    if(service == null) { service = new ServiceImpl(); }
```

This is NOT ThreadSafe

```
    return service.work(txt);
  }
}
```

Virtual Proxy

@SvenRuppert

```
public class VirtualService implements Service {
  private Service service = null;
  public String work(String txt) {
    if(service == null) { service = new ServiceImpl(); }
```

This is NOT ThreadSafe

```
    return service.work(txt);
  }
}
```

Virtual Proxy

@SvenRuppert

```
public class VirtualService implements Service {
  private Service service = null;
  public String work(String txt) {
    if(service == null) { service = new ServiceImpl(); }
```

This is NOT ThreadSafe

fixed decision for an implementation

```
    return service.work(txt);
  }
}
```

Virtual Proxy

@SvenRuppert

```
public class VirtualService implements Service {
  private Service service = null;
  public String work(String txt) {
    if(service == null) { service = new ServiceImpl(); }

    return service.work(txt);
  }
}
```

This is NOT ThreadSafe

fixed decision for an implementation

Virtual Proxy

@SvenRuppert

```
public class VirtualService implements Service {
  private Service service = null;
  public String work(String txt) {
    if(service == null) { service = new ServiceImpl(); }
```

This is NOT ThreadSafe

fixed decision for an implementation

```
    return service.work(txt);
  }
}
```

how to combine it with a FactoryPattern ?

Virtual Proxy

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

Virtual Proxy

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

Virtual Proxy

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
public interface ServiceFactory {
  Service createInstance();
}
```

Virtual Proxy

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
public interface ServiceFactory {
  Service createInstance();
}
```

Virtual Proxy

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
public interface ServiceFactory {
  Service createInstance();
}
```

```
public interface ServiceStrategyFactory {
   Service realSubject(ServiceFactory factory);
}
```

Virtual Proxy - Not - ThreadSafe

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

Virtual Proxy - Not - ThreadSafe

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

Virtual Proxy - Not - ThreadSafe

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
private ServiceFactory serviceFactory = ServiceImpl::new;
```

Virtual Proxy - Not - ThreadSafe

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
private ServiceFactory serviceFactory = ServiceImpl::new;
```

Virtual Proxy - Not - ThreadSafe

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
private ServiceFactory serviceFactory = ServiceImpl::new;
```

```
public class ServiceStrategyFactoryImpl implements ServiceStrategyFactory {
    Service realSubject;
    public Service realSubject(final ServiceFactory factory) {
        if (realSubject == null) {
            realSubject = factory.createInstance();
        }
        return realSubject;
    }
}
```

Virtual Proxy - Not - ThreadSafe

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

Virtual Proxy - Not - ThreadSafe


```
if(service == null) { service = new ServiceImpl(); }
```

```
public class ServiceProxy implements Service {
```

Virtual Proxy - Not - ThreadSafe

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
public class ServiceProxy implements Service {
  private ServiceFactory serviceFactory = ServiceImpl::new;
```

Virtual Proxy - Not - ThreadSafe

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
public class ServiceProxy implements Service {
  private ServiceFactory serviceFactory = ServiceImpl::new;
  private ServiceStrategyFactory strategyFactory = new ServiceStrategyFactoryImpl();
```

Virtual Proxy - Not - ThreadSafe

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
public class ServiceProxy implements Service {
  private ServiceFactory serviceFactory = ServiceImpl::new;
  private ServiceStrategyFactory strategyFactory = new ServiceStrategyFactoryImpl();

  public String work(String txt) {
```

Virtual Proxy - Not - ThreadSafe


```
if(service == null) { service = new ServiceImpl(); }
```

```
public class ServiceProxy implements Service {
  private ServiceFactory serviceFactory = ServiceImpl::new;
  private ServiceStrategyFactory strategyFactory = new ServiceStrategyFactoryImpl();

  public String work(String txt) {

  }
}
```

Virtual Proxy - Not - ThreadSafe

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
public class ServiceProxy implements Service {
  private ServiceFactory serviceFactory = ServiceImpl::new;
  private ServiceStrategyFactory strategyFactory = new ServiceStrategyFactoryImpl();

  public String work(String txt) {
    return
  }
}
```

Virtual Proxy - Not - ThreadSafe

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
public class ServiceProxy implements Service {
  private ServiceFactory serviceFactory = ServiceImpl::new;
  private ServiceStrategyFactory strategyFactory = new ServiceStrategyFactoryImpl();

  public String work(String txt) {
    return strategyFactory
  }
}
```

Virtual Proxy - Not - ThreadSafe

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
public class ServiceProxy implements Service {
  private ServiceFactory serviceFactory = ServiceImpl::new;
  private ServiceStrategyFactory strategyFactory = new ServiceStrategyFactoryImpl();

  public String work(String txt) {
    return strategyFactory .realSubject(
  }
}
```

Virtual Proxy - Not - ThreadSafe

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
public class ServiceProxy implements Service {
  private ServiceFactory serviceFactory = ServiceImpl::new;
  private ServiceStrategyFactory strategyFactory = new ServiceStrategyFactoryImpl();

  public String work(String txt) {
    return strategyFactory .realSubject(serviceFactory
  }
}
```

Virtual Proxy - Not - ThreadSafe

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
public class ServiceProxy implements Service {
  private ServiceFactory serviceFactory = ServiceImpl::new;
  private ServiceStrategyFactory strategyFactory = new ServiceStrategyFactoryImpl();

  public String work(String txt) {
    return strategyFactory .realSubject(serviceFactory).work(txt);
  }
}
```

Virtual Proxy - Synchronized


```
if(service == null) { service = new ServiceImpl(); }
```

Virtual Proxy - Synchronized

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

Virtual Proxy - Synchronized

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
private ServiceFactory serviceFactory = ServiceImpl::new;
```

Virtual Proxy - Synchronized

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
private ServiceFactory serviceFactory = ServiceImpl::new;
```

Virtual Proxy - Synchronized

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
private ServiceFactory serviceFactory = ServiceImpl::new;
```

```
public class ServiceStrategyFactorySynchronized implements ServiceStrategyFactory {
  Service realSubject;
  public synchronized Service realSubject(final ServiceFactory factory) {
    if (realSubject == null) {
      realSubject = factory.createInstance();
    }
    return realSubject;
  }
}
```

Virtual Proxy - Synchronized

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
private ServiceFactory serviceFactory = ServiceImpl::new;
```

```
public class ServiceStrategyFactorySynchronized implements ServiceStrategyFactory {
  Service realSubject;
  public synchronized Service realSubject(final ServiceFactory factory) {
    if (realSubject == null) {
      realSubject = factory.createInstance();
    }
    return realSubject;
  }
}
```

This is ThreadSafe

Virtual Proxy - Method Scoped

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

Virtual Proxy - Method Scoped

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

Virtual Proxy - Method Scoped

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
private ServiceFactory serviceFactory = ServiceImpl::new;
```

Virtual Proxy - Method Scoped

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
private ServiceFactory serviceFactory = ServiceImpl::new;
```

Virtual Proxy - Method Scoped

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
private ServiceFactory serviceFactory = ServiceImpl::new;
```

```
public class ServiceStrategyFactoryMethodScoped implements
ServiceStrategyFactory {
  public Service realSubject(final ServiceFactory factory) {
    return factory.createInstance();
  }
}
```

Virtual Proxy - Method Scoped

@SvenRuppert

```
if(service == null) { service = new ServiceImpl(); }
```

```
private ServiceFactory serviceFactory = ServiceImpl::new;
```

```
public class ServiceStrategyFactoryMethodScoped implements
ServiceStrategyFactory {
  public Service realSubject(final ServiceFactory factory) {
    return factory.createInstance();
  }
}
```

Virtual Proxy - Show some code….

@SvenRuppert

DynamicObjectAdapter - orig Version

@SvenRuppert

the original Version of the DynamicObjectAdapter

was written by Dr. Heinz Kabutz

DynamicObjectAdapter - orig Version

@SvenRuppert

DynamicObjectAdapter - orig Version

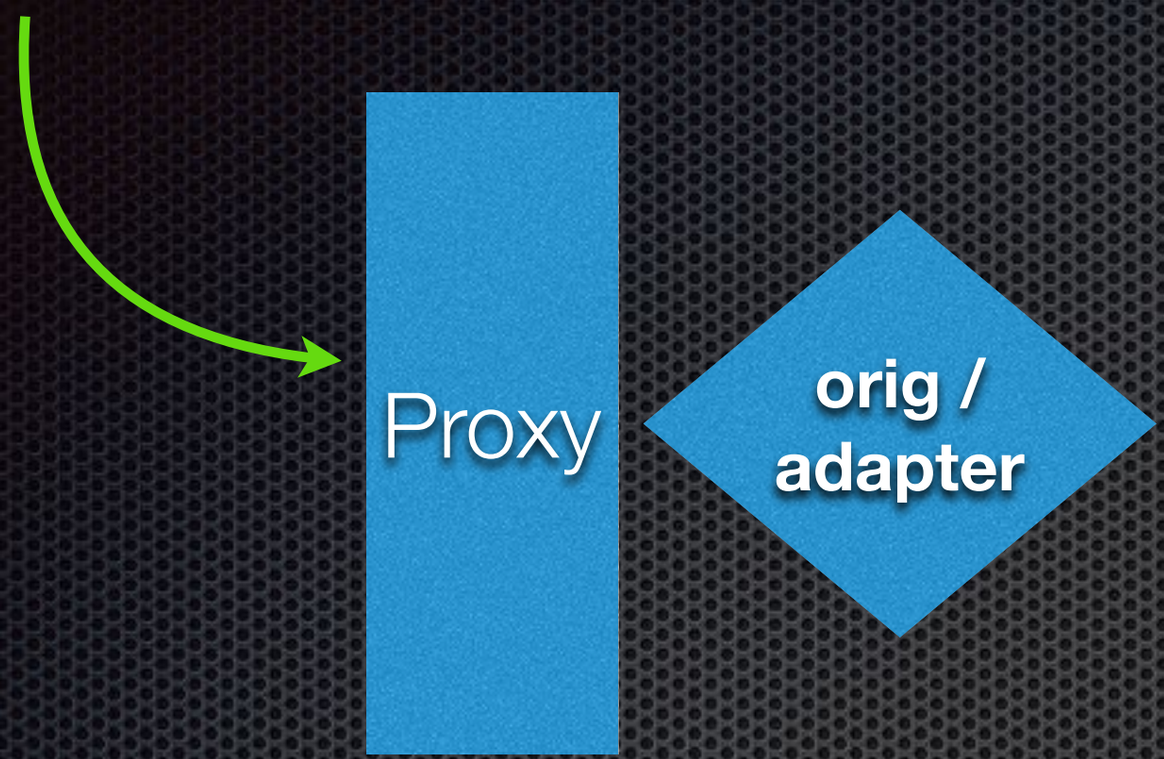
@SvenRuppert

DynamicObjectAdapter - orig Version

@SvenRuppert

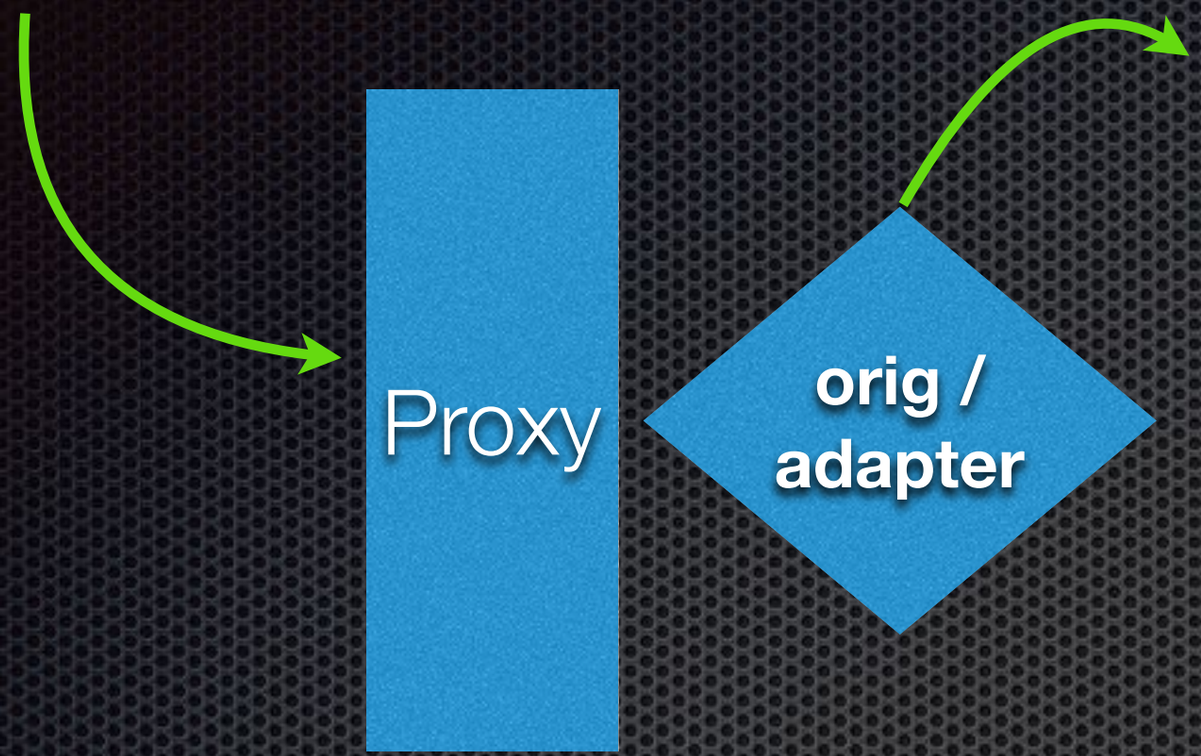
DynamicObjectAdapter - orig Version

@SvenRuppert



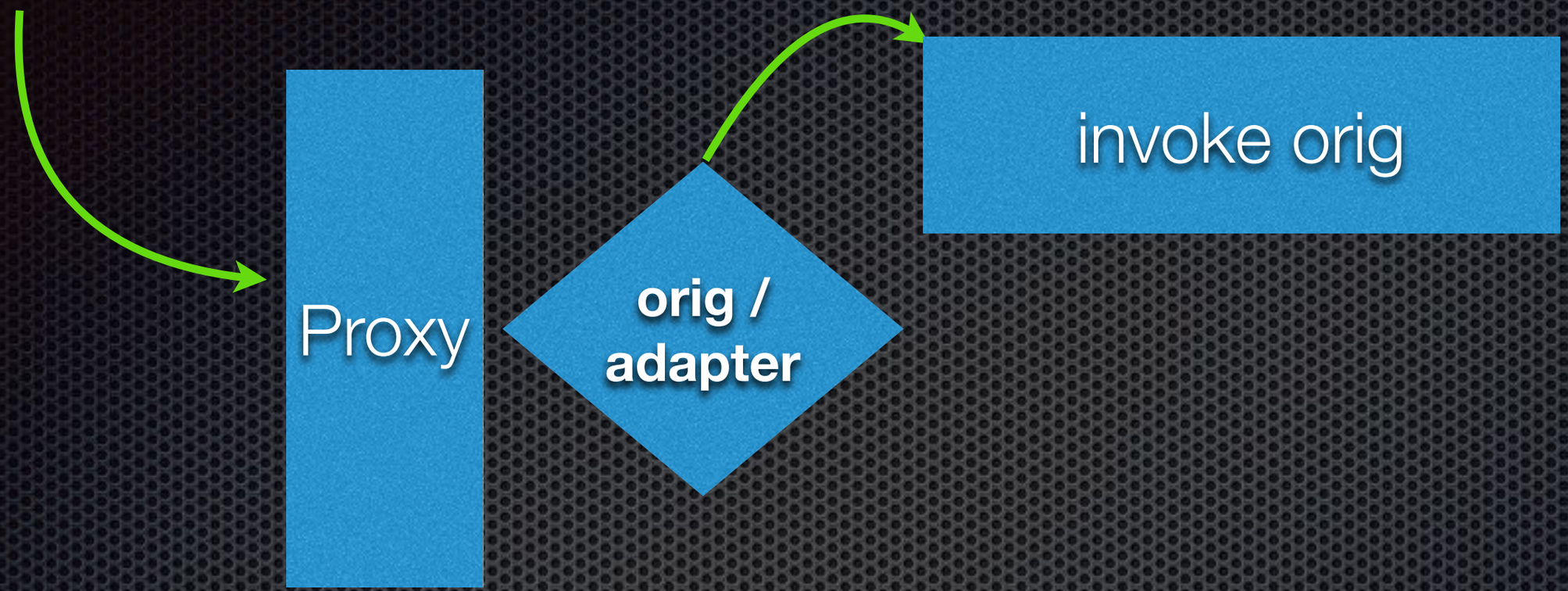
DynamicObjectAdapter - orig Version

@SvenRuppert



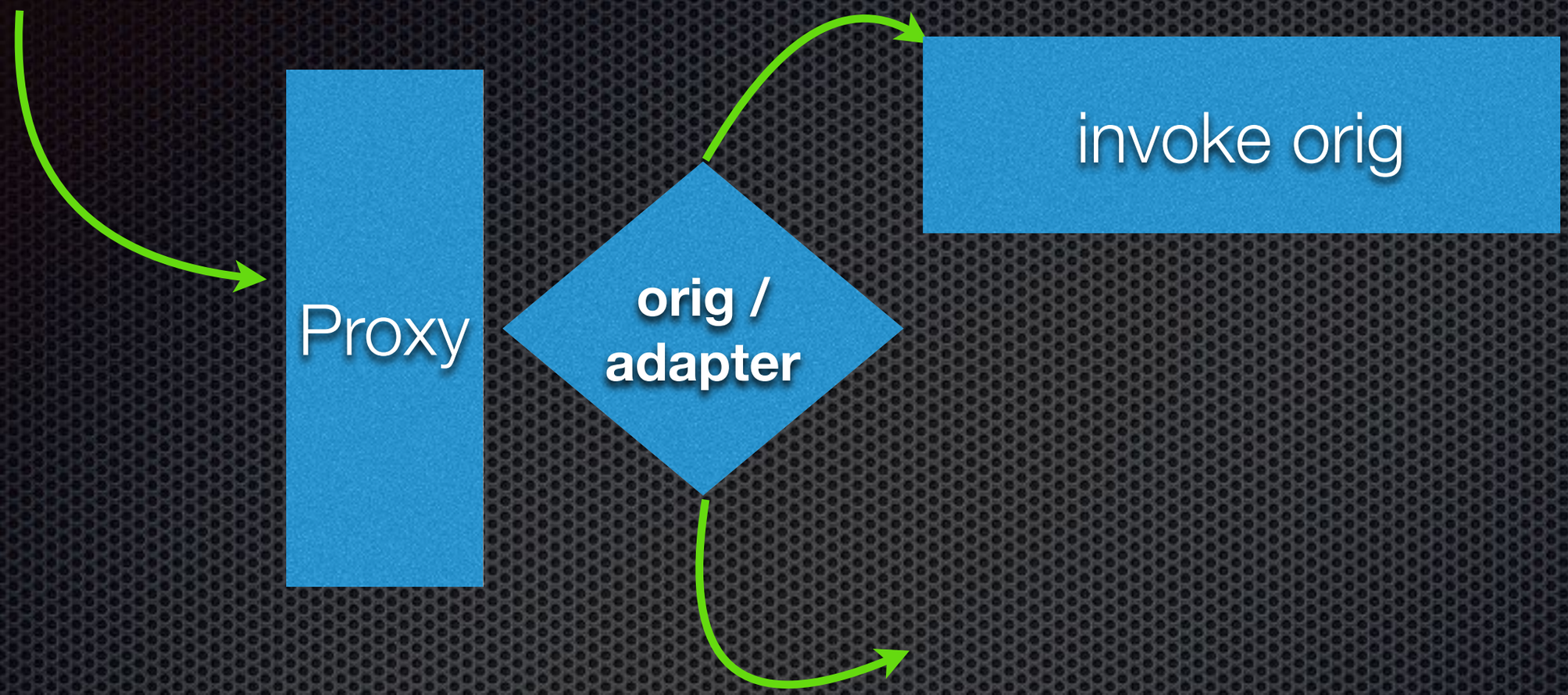
DynamicObjectAdapter - orig Version

@SvenRuppert



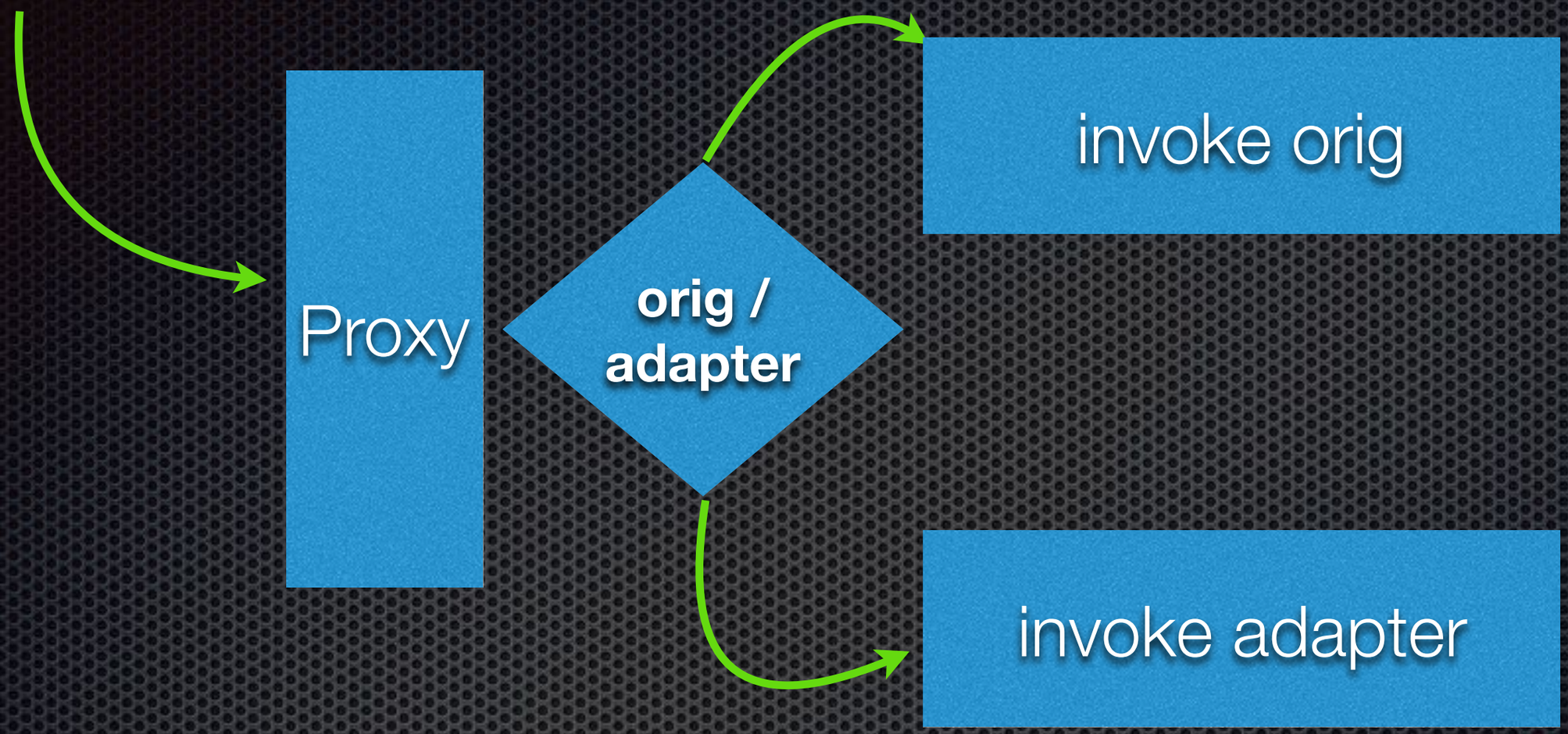
DynamicObjectAdapter - orig Version

@SvenRuppert



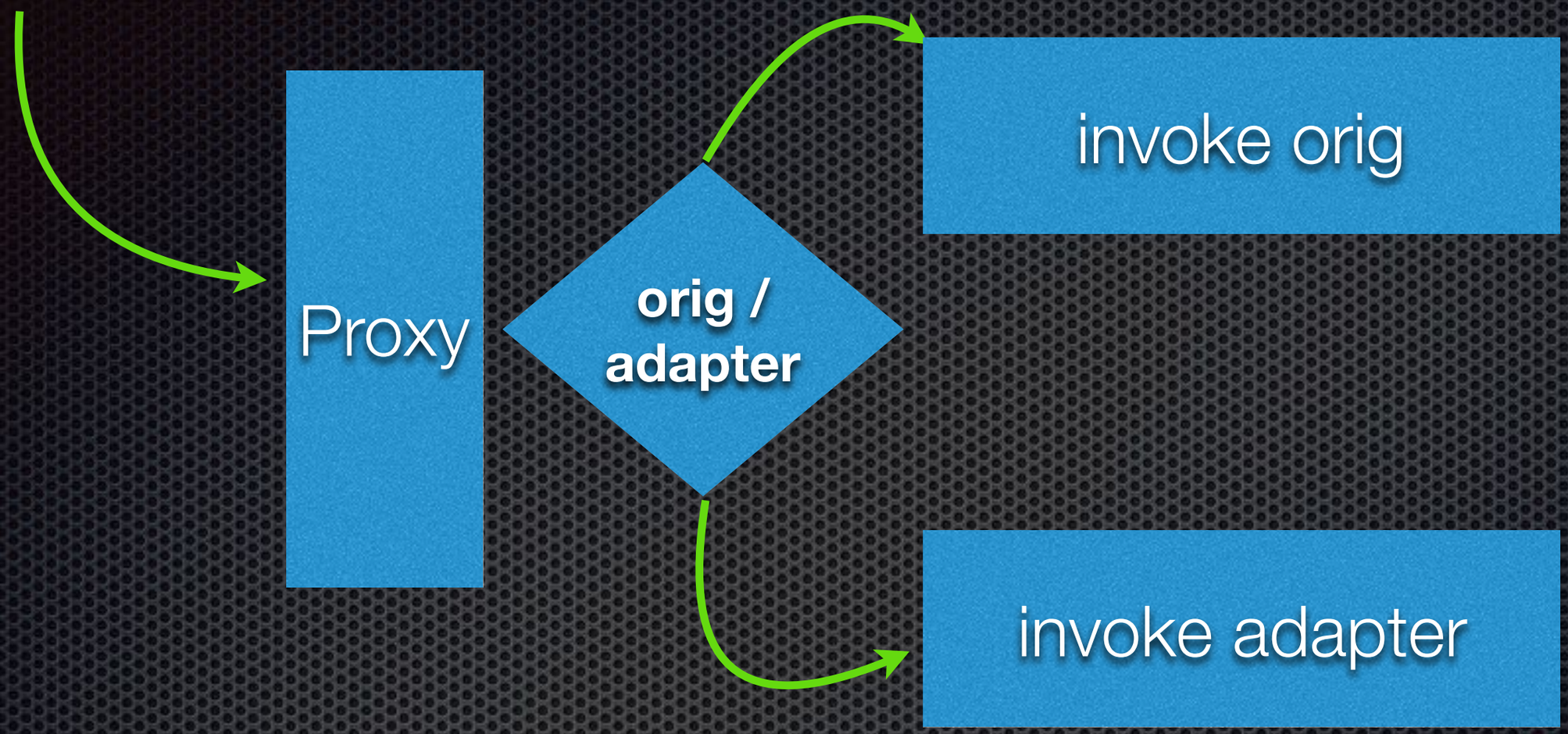
DynamicObjectAdapter - orig Version

@SvenRuppert



DynamicObjectAdapter - orig Version

@SvenRuppert



Goal: do not need to write an Adapter with Delegator

DynamicObjectAdapter - orig Version

@SvenRuppert

core concept:
switching between method implementations

DynamicObjectAdapter - orig Version

@SvenRuppert

core concept:
switching between method implementations

how to identify a method?

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
public interface Service {
  String doWork_A();
}

public static class ServiceImpl implements Service {
  public String doWork_A() {
    return ServiceImpl.class.getSimpleName();
  }
}
```

DynamicObjectAdapter - orig Version

how to identify a method?

```
public class MethodIdentifier {
    private final String name;
    private final Class[] parameters;

    public MethodIdentifier(Method m) {
        name = m.getName();
        parameters = m.getParameterTypes();
    }
      // we can save time by assuming that we only compare against
    // other MethodIdentifier objects
    public boolean equals(Object o) {
        MethodIdentifier mid = (MethodIdentifier) o;
        return name.equals(mid.name) &&
                Arrays.equals(parameters, mid.parameters);
    }
    public int hashCode() { return name.hashCode(); }
}
```

DynamicObjectAdapter - orig Version

how to identify a method?

```
public class MethodIdentifier {
    private final String name;
    private final Class[] parameters;

    public MethodIdentifier(Method m) {
        name = m.getName();
        parameters = m.getParameterTypes();
    }
      // we can save time by assuming that we only compare against
    // other MethodIdentifier objects
    public boolean equals(Object o) {
        MethodIdentifier mid = (MethodIdentifier) o;
        return name.equals(mid.name) &&
                Arrays.equals(parameters, mid.parameters);
    }
    public int hashCode() { return name.hashCode(); }
}
```

DynamicObjectAdapter - orig Version

how to identify a method?

```
public class MethodIdentifier {
    private final String name;
    private final Class[] parameters;

    public MethodIdentifier(Method m) {
        name = m.getName();
        parameters = m.getParameterTypes();
    }
      // we can save time by assuming that we only compare against
    // other MethodIdentifier objects
    public boolean equals(Object o) {
        MethodIdentifier mid = (MethodIdentifier) o;
        return name.equals(mid.name) &&
                Arrays.equals(parameters, mid.parameters);
    }
    public int hashCode() { return name.hashCode(); }
}
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
  private T original;
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
  private T original;    private Object adapter;
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
  private T original;    private Object adapter;
  private void addAdapter(Object adapter) {
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
     this.adapter = adapter;
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
     this.adapter = adapter;
     final Class<?> adapterClass = adapter.getClass();
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
     this.adapter = adapter;
     final Class<?> adapterClass = adapter.getClass();
     Method[] methods = adapterClass.getDeclaredMethods();
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
     this.adapter = adapter;
     final Class<?> adapterClass = adapter.getClass();
     Method[] methods = adapterClass.getDeclaredMethods();
     for (Method m : methods) {
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
     this.adapter = adapter;
     final Class<?> adapterClass = adapter.getClass();
     Method[] methods = adapterClass.getDeclaredMethods();
     for (Method m : methods) {
        final MethodIdentifier key = new MethodIdentifier(m);
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
     this.adapter = adapter;
     final Class<?> adapterClass = adapter.getClass();
     Method[] methods = adapterClass.getDeclaredMethods();
     for (Method m : methods) {
        final MethodIdentifier key = new MethodIdentifier(m);
        adaptedMethods.put(key, m);
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
     this.adapter = adapter;
     final Class<?> adapterClass = adapter.getClass();
     Method[] methods = adapterClass.getDeclaredMethods();
     for (Method m : methods) {
       final MethodIdentifier key = new MethodIdentifier(m);
       adaptedMethods.put(key, m);
  }
}
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
     this.adapter = adapter;
     final Class<?> adapterClass = adapter.getClass();
     Method[] methods = adapterClass.getDeclaredMethods();
     for (Method m : methods) {
        final MethodIdentifier key = new MethodIdentifier(m);
        adaptedMethods.put(key, m);
     }
  }
```

```
public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
     this.adapter = adapter;
     final Class<?> adapterClass = adapter.getClass();
     Method[] methods = adapterClass.getDeclaredMethods();
     for (Method m : methods) {
        final MethodIdentifier key = new MethodIdentifier(m);
         adaptedMethods.put(key, m);
     }
  }
```

```
public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
//SNIPP
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
     this.adapter = adapter;
     final Class<?> adapterClass = adapter.getClass();
     Method[] methods = adapterClass.getDeclaredMethods();
     for (Method m : methods) {
        final MethodIdentifier key = new MethodIdentifier(m);
         adaptedMethods.put(key, m);
     }
    }




  public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
  //SNIPP
  }
}
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
    //SNIPP
  }



  public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {




  }
}
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
    //SNIPP
  }



  public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
    try {
      final MethodIdentifier key = new MethodIdentifier(m);




  }
}
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
    //SNIPP
  }



  public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
    try {
      final MethodIdentifier key = new MethodIdentifier(m);
      Method other = adaptedMethods.get(key);



  }
}
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
    //SNIPP
  }


  public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
    try {
      final MethodIdentifier key = new MethodIdentifier(m);
      Method other = adaptedMethods.get(key);
       if (other != null) { return other.invoke(adapter, args);  }


  }
}
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
    //SNIPP
  }


  public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
    try {
      final MethodIdentifier key = new MethodIdentifier(m);
      Method other = adaptedMethods.get(key);
       if (other != null) { return other.invoke(adapter, args);  }
       else                { return      m.invoke(original, args);   }

  }
}
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
    //SNIPP
  }


  public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
    try {
      final MethodIdentifier key = new MethodIdentifier(m);
      Method other = adaptedMethods.get(key);
      if (other != null) { return other.invoke(adapter, args);  }
      else               { return      m.invoke(original, args);   }
    } catch (InvocationTargetException e) { throw e.getTargetException(); }
  }
}
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
    //SNIPP
  }
```

what could be a problem ?

```
  public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
    try {
      final MethodIdentifier key = new MethodIdentifier(m);
      Method other = adaptedMethods.get(key);
      if (other != null) { return other.invoke(adapter, args);  }
      else               { return      m.invoke(original, args);   }
    } catch (InvocationTargetException e) { throw e.getTargetException(); }
  }
}
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private static class DOAInvocationHandler<T extends I, I> implements InvocationHandler {
  private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

  private T original;    private Object adapter;

  private void addAdapter(Object adapter) {
    //SNIPP
  }
```

what could be a problem ?

what could be improved ?

```
  public Object invoke(Object proxy, Method m, Object[] args) throws Throwable {
    try {
      final MethodIdentifier key = new MethodIdentifier(m);
      Method other = adaptedMethods.get(key);
      if (other != null) { return other.invoke(adapter, args);  }
      else               { return      m.invoke(original, args);   }
    } catch (InvocationTargetException e) { throw e.getTargetException(); }
  }
}
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

private T original;    private Object adapter;

private void addAdapter(Object adapter) {
```

what could be a problem ?

what could be improved ?

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

private T original;    private Object adapter;

private void addAdapter(Object adapter) {
```

what could be a problem ?

what could be improved ?

```
public interface Service {
  String doWork_A();
}
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

private T original;    private Object adapter;

private void addAdapter(Object adapter) {
```

what could be a problem ?

what could be improved ?

```
  public interface Service {
    String doWork_A();
  }

  public class Adapter_A {
    public String doWork_A() { .. }
  }
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

private T original;    private Object adapter;

private void addAdapter(Object adapter) {
```

what could be a problem ?

what could be improved ?

```
public interface Service {
  String doWork_A();
}
```

```
public class Adapter_A {
  public String doWork_A() { .. }
}
```

```
public class Adapter_B {
  public String doWork_B() { .. }
}
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

private T original;    private Object adapter;

private void addAdapter(Object adapter) {
```

what could be a problem ?

what could be improved ?

```
public interface Service {
  String doWork_A();
}

public class Adapter_A {
  public String doWork_A() { .. }
}

public class Adapter_B {
  public String doWork_B() { .. }
}
```

DynamicObjectAdapter - orig. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();

private T original;    private Object adapter;

private void addAdapter(Object adapter) {
```

what could be a problem ?

what could be improved ?

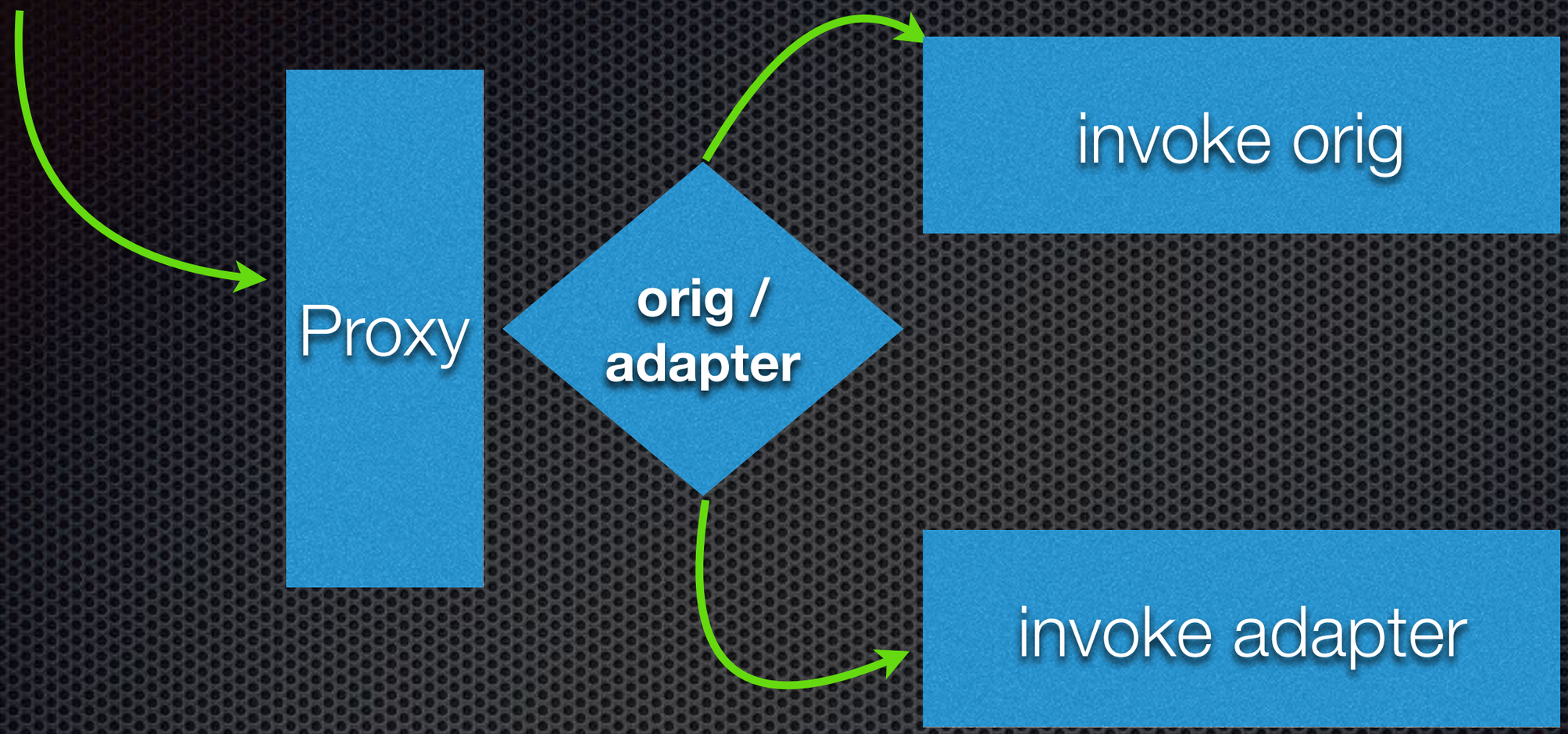
```
public interface Service {
  String doWork_A();
}
```

```
public class Adapter_A {
  public String doWork_A() { .. }
}
```

```
public class Adapter_B {
  public String doWork_B() { .. }
}
```

DynamicObjectAdapter - orig Version

@SvenRuppert



DynamicObjectAdapter - ext. Version

@SvenRuppert

first extension !

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
public interface Service {
  String doWork_A(String txt);
  String doWork_B(String txt);
}

public static class ServiceImpl implements Service {

  public String doWork_A(String txt) {
    return "doWorkd_A_Original";
  }

  public String doWork_B(String txt) {
    return "doWorkd_B_Original";
  }
}
```

DynamicObjectAdapter - ext. Version

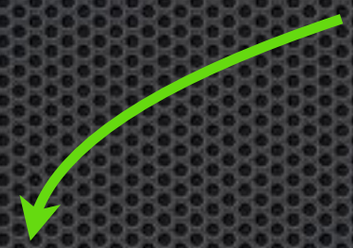
@SvenRuppert

```
public interface Service {
  String doWork_A(String txt);
  String doWork_B(String txt);
}

public interface ServiceAdapter_A {
  String doWork_A(String txt);
}

public interface ServiceAdapter_B {
  String doWork_B(String txt);
}
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
public interface Service {
  String doWork_A(String txt);
  String doWork_B(String txt);
}
```



```
public interface ServiceAdapter_A {
  String doWork_A(String txt);
}

public interface ServiceAdapter_B {
  String doWork_B(String txt);
}
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
public interface Service {
  String doWork_A(String txt);
  String doWork_B(String txt);
}
```

functional interface

```
public interface ServiceAdapter_A {
  String doWork_A(String txt);
}

public interface ServiceAdapter_B {
  String doWork_B(String txt);
}
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
public interface Service {
  String doWork_A(String txt);
  String doWork_B(String txt);
}
```

functional interface

```
public interface ServiceAdapter_A {
  String doWork_A(String txt);
}

public interface ServiceAdapter_B {
  String doWork_B(String txt);
}
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
public interface Service {
  String doWork_A(String txt);
  String doWork_B(String txt);
}

public interface ServiceAdapter_A {
  String doWork_A(String txt);
}

public interface ServiceAdapter_B {
  String doWork_B(String txt);
}
```

functional interface

functional interface

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;

private void addAdapter(Object adapter) {
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;

private void addAdapter(Object adapter) {
  final Class<?> adapterClass = adapter.getClass();
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;

private void addAdapter(Object adapter) {
  final Class<?> adapterClass = adapter.getClass();
  Method[] methods = adapterClass.getDeclaredMethods();
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;

private void addAdapter(Object adapter) {
  final Class<?> adapterClass = adapter.getClass();
  Method[] methods = adapterClass.getDeclaredMethods();
  for (Method m : methods) {
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;

private void addAdapter(Object adapter) {
  final Class<?> adapterClass = adapter.getClass();
  Method[] methods = adapterClass.getDeclaredMethods();
  for (Method m : methods) {
    final MethodIdentifier key = new MethodIdentifier(m);
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;

private void addAdapter(Object adapter) {
  final Class<?> adapterClass = adapter.getClass();
  Method[] methods = adapterClass.getDeclaredMethods();
  for (Method m : methods) {
    final MethodIdentifier key = new MethodIdentifier(m);
    adaptedMethods.put(key, m);
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;

private void addAdapter(Object adapter) {
  final Class<?> adapterClass = adapter.getClass();
  Method[] methods = adapterClass.getDeclaredMethods();
  for (Method m : methods) {
    final MethodIdentifier key = new MethodIdentifier(m);
    adaptedMethods.put(key, m);
    adapters.put(key, adapter);
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;

private void addAdapter(Object adapter) {
  final Class<?> adapterClass = adapter.getClass();
  Method[] methods = adapterClass.getDeclaredMethods();
  for (Method m : methods) {
    final MethodIdentifier key = new MethodIdentifier(m);
    adaptedMethods.put(key, m);
    adapters.put(key, adapter);
  }
}
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;

public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;

public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
try {
    final MethodIdentifier key = new MethodIdentifier(method);
    Method other = adaptedMethods.get(key);
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;

public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
try {
  final MethodIdentifier key = new MethodIdentifier(method);
  Method other = adaptedMethods.get(key);

  if (other != null) {
    return other.invoke(adapters.get(key), args);
  } else {
    return method.invoke(original, args);
  }
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;

public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
try {
  final MethodIdentifier key = new MethodIdentifier(method);
  Method other = adaptedMethods.get(key);

  if (other != null) {
    return other.invoke(adapters.get(key), args);
   } else {
    return method.invoke(original, args);
   }
 } catch (InvocationTargetException e) {
   throw e.getTargetException();
 }
}
```

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;
```

if you will use this without a Builder
you can add useless Adapter !!

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;
```

if you will use this without a Builder
you can add useless Adapter !!

DynamicObjectAdapter - ext. Version

@SvenRuppert

```
private Map<MethodIdentifier, Method> adaptedMethods = new HashMap<>();
private Map<MethodIdentifier, Object> adapters = new HashMap<>();

private T original;
private Class<T> target;
```

if you will use this without a Builder
you can add useless Adapter !!

Add some typed with-Methods

DynamicObjectAdapter - orig Version

@SvenRuppert

Add some typed with-Methods

```
public AdapterBuilder<T> withDoWork_A(ServiceAdapter_A adapter) {
  addAdapter(adapter);
  return this;
}
```

DynamicObjectAdapter - orig Version

@SvenRuppert

Add some typed with-Methods

```
public AdapterBuilder<T> withDoWork_A(ServiceAdapter_A adapter) {
  addAdapter(adapter);
  return this;
}
```

remember

DynamicObjectAdapter - orig Version

@SvenRuppert

Add some typed with-Methods

```
public AdapterBuilder<T> withDoWork_A(ServiceAdapter_A adapter) {
  addAdapter(adapter);
  return this;
}
```

remember

```
public interface Service {
  String doWork_A(String txt);
  String doWork_B(String txt);
}
```

DynamicObjectAdapter - orig Version

@SvenRuppert

Add some typed with-Methods

```
public AdapterBuilder<T> withDoWork_A(ServiceAdapter_A adapter) {
  addAdapter(adapter);
  return this;
}
```

remember

```
public interface Service {
  String doWork_A(String txt);
  String doWork_B(String txt);
}

public interface ServiceAdapter_A {
  String doWork_A(String txt);
}
```

DynamicObjectAdapter - orig Version

How to use this?

DynamicObjectAdapter - orig Version

How to use this?

```
Service service = AdapterBuilder.<Service>newBuilder()
```

DynamicObjectAdapter - orig Version

How to use this?

```
Service service = AdapterBuilder.<Service>newBuilder()
        .withTarget(Service.class)
```

DynamicObjectAdapter - orig Version

How to use this?

```
Service service = AdapterBuilder.<Service>newBuilder()
        .withTarget(Service.class)
        .withOriginal(new ServiceImpl())
```

DynamicObjectAdapter - orig Version

@SvenRuppert

How to use this?

```
Service service = AdapterBuilder.<Service>newBuilder()
        .withTarget(Service.class)
        .withOriginal(new ServiceImpl())
        .withDoWork_A((txt) -> txt + "_part")
```

DynamicObjectAdapter - orig Version

@SvenRuppert

How to use this?

```
Service service = AdapterBuilder.<Service>newBuilder()
        .withTarget(Service.class)
        .withOriginal(new ServiceImpl())
        .withDoWork_A((txt) -> txt + "_part")
        .build();
```

DynamicObjectAdapter - orig Version

How to use this like a mock ?

DynamicObjectAdapter - orig Version

@SvenRuppert

How to use this like a mock ?

```
Service service = AdapterBuilder.<Service>newBuilder()
```

DynamicObjectAdapter - orig Version

@SvenRuppert

How to use this like a mock ?

```
Service service = AdapterBuilder.<Service>newBuilder()
        .withTarget(Service.class)
```

DynamicObjectAdapter - orig Version

@SvenRuppert

How to use this like a mock ?

```
Service service = AdapterBuilder.<Service>newBuilder()
        .withTarget(Service.class)
        .withOriginal(null)
```

DynamicObjectAdapter - orig Version

@SvenRuppert

How to use this like a mock ?

```
Service service = AdapterBuilder.<Service>newBuilder()
        .withTarget(Service.class)
        .withOriginal(null)
        .withDoWork_A((txt) -> txt + "_part")
```

DynamicObjectAdapter - orig Version

@SvenRuppert

How to use this like a mock ?

```
Service service = AdapterBuilder.<Service>newBuilder()
        .withTarget(Service.class)
        .withOriginal(null)
        .withDoWork_A((txt) -> txt + "_part")
        .build();
```

generated DynamicObjectAdapterBuilder @SvenRuppert

To much boring code to write

but we want to have a type-safe version

if wrong… compile must break

generated DynamicObjectAdapterBuilder @SvenRuppert

To much boring code to write

but we want to have a type-safe version

if wrong… compile must break

What could we change now ?

generated DynamicObjectAdapterBuilder @SvenRuppert

remember

generated DynamicObjectAdapterBuilder @SvenRuppert

remember

```
public interface Service {
  String doWork_A(String txt);
  String doWork_B(String txt);
}
```

generated DynamicObjectAdapterBuilder @SvenRuppert

remember

```
public interface Service {
  String doWork_A(String txt);
  String doWork_B(String txt);
}
public interface ServiceAdapter_A {
  String doWork_A(String txt);
}

public interface ServiceAdapter_B {
  String doWork_B(String txt);
}
```

generated DynamicObjectAdapterBuilder @SvenRuppert

remember

```
public interface Service {
  String doWork_A(String txt);
  String doWork_B(String txt);
}
public interface ServiceAdapter_A {
  String doWork_A(String txt);
}

public interface ServiceAdapter_B {
  String doWork_B(String txt);
}
```

generated DynamicObjectAdapterBuilder @SvenRuppert

remember

```
public interface Service {
  String doWork_A(String txt);
  String doWork_B(String txt);
 }
public interface ServiceAdapter_A {
  String doWork_A(String txt);
}

public interface ServiceAdapter_B {
  String doWork_B(String txt);
}
```

functional interface

generated DynamicObjectAdapterBuilder @SvenRuppert

use Annotation Processing

generated DynamicObjectAdapterBuilder

@SvenRuppert

use Annotation Processing

create a marker like
@DynamicObjectAdapterBuilder

generated DynamicObjectAdapterBuilder

@SvenRuppert

use Annotation Processing

create a marker like
@DynamicObjectAdapterBuilder

```
@Retention(RetentionPolicy.SOURCE)
@Target(ElementType.TYPE)
public @interface DynamicObjectAdapterBuilder {
}
```

generated DynamicObjectAdapterBuilder @SvenRuppert

use Annotation Processing

create a marker like
@DynamicObjectAdapterBuilder

```
@Retention(RetentionPolicy.SOURCE)
@Target(ElementType.TYPE)
public @interface DynamicObjectAdapterBuilder {
}

public class AnnotationProcessor extends AbstractProcessor
```

generated DynamicObjectAdapterBuilder @SvenRuppert

use Annotation Processing

with every compile you will get actual

generated DynamicObjectAdapterBuilder @SvenRuppert

use Annotation Processing

with every compile you will get actual

Functional-Interfaces for the Adapters

generated DynamicObjectAdapterBuilder @SvenRuppert

use Annotation Processing

with every compile you will get actual

Functional-Interfaces for the Adapters

generated AdapterBuilder

StaticObjectAdapter

to avoid the latency of the dynamic Proxy

we could implement **static ObjectAdapter**

the basic idea is the same as we had with the **DynamicObjectAdapter**

StaticObjectAdapter

split the interface into Functional Interfaces

create for every Functional Interface an attribute

at runtime: if an attribute
is available invoke the method .....

otherwise use the delegator

StaticObjectAdapter


```
public interface Service {
  String doWorkA(String txt);
  String doWorkB(String txt);
}
```

StaticObjectAdapter

@SvenRuppert

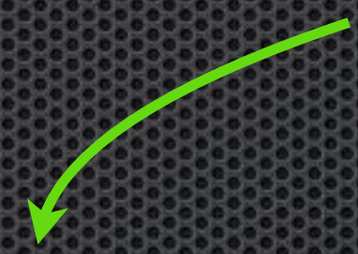
```
public interface Service {
  String doWorkA(String txt);
  String doWorkB(String txt);
}
```

```
public interface Service_DoWorkA_Adapter{
  String doWorkA(final String txt);
}
public interface Service_DoWorkB_Adapter{
  String doWorkB(final String txt);
}
```

StaticObjectAdapter

@SvenRuppert

```
public interface Service {
  String doWorkA(String txt);
  String doWorkB(String txt);
}
```



```
public interface Service_DoWorkA_Adapter{
  String doWorkA(final String txt);
}
public interface Service_DoWorkB_Adapter{
  String doWorkB(final String txt);
}
```

StaticObjectAdapter

@SvenRuppert

```
public interface Service {
  String doWorkA(String txt);
  String doWorkB(String txt);
}
```

functional interface

```
public interface Service_DoWorkA_Adapter{
  String doWorkA(final String txt);
}
public interface Service_DoWorkB_Adapter{
  String doWorkB(final String txt);
}
```

StaticObjectAdapter

StaticObjectAdapter

@SvenRuppert

```
public class ServiceStaticObjectAdapter implements Service {
```

StaticObjectAdapter

@SvenRuppert

```
public class ServiceStaticObjectAdapter implements Service {
  private Service service;
```

StaticObjectAdapter

@SvenRuppert

```
public class ServiceStaticObjectAdapter implements Service {
  private Service service;
  private Service_DoWorkA_Adapter doWorkAAdapter;
```

StaticObjectAdapter

@SvenRuppert

```
public class ServiceStaticObjectAdapter implements Service {

  private Service service;
  private Service_DoWorkA_Adapter doWorkAAdapter;
  private Service_DoWorkB_Adapter doWorkBAdapter;
```

StaticObjectAdapter

@SvenRuppert

```
public class ServiceStaticObjectAdapter implements Service {

  private Service service;
  private Service_DoWorkA_Adapter doWorkAAdapter;
  private Service_DoWorkB_Adapter doWorkBAdapter;

  public String doWorkA(final String txt) {



  }
```

StaticObjectAdapter

@SvenRuppert

```
public class ServiceStaticObjectAdapter implements Service {

  private Service service;
  private Service_DoWorkA_Adapter doWorkAAdapter;
  private Service_DoWorkB_Adapter doWorkBAdapter;

  public String doWorkA(final String txt) {
    if (this.doWorkAAdapter != null)

  }
```

StaticObjectAdapter

@SvenRuppert

```
public class ServiceStaticObjectAdapter implements Service {

  private Service service;
  private Service_DoWorkA_Adapter doWorkAAdapter;
  private Service_DoWorkB_Adapter doWorkBAdapter;

  public String doWorkA(final String txt) {
    if (this.doWorkAAdapter != null)    return this.doWorkAAdapter.doWorkA(txt);

  }
```

StaticObjectAdapter

@SvenRuppert

```
public class ServiceStaticObjectAdapter implements Service {

  private Service service;
  private Service_DoWorkA_Adapter doWorkAAdapter;
  private Service_DoWorkB_Adapter doWorkBAdapter;

  public String doWorkA(final String txt) {
    if (this.doWorkAAdapter != null)    return this.doWorkAAdapter.doWorkA(txt);
                                        return service.doWorkA(txt)
  }
```

StaticObjectAdapter

@SvenRuppert

```
public class ServiceStaticObjectAdapter implements Service {

  private Service service;
  private Service_DoWorkA_Adapter doWorkAAdapter;
  private Service_DoWorkB_Adapter doWorkBAdapter;

  public String doWorkA(final String txt) {
    if (this.doWorkAAdapter != null)    return this.doWorkAAdapter.doWorkA(txt);
                                        return service.doWorkA(txt)
  }
```

We can generate this !

generated StaticObjectAdapter

with AnnotationProcessing

- split the interface with **@StaticObjectAdapter**
  - write functional interfaces
    - generate the StaticObjectAdapter
    - mark the Adapter with **@IsObjectAdapter**

Decorator - pre Java8

Decorator - pre Java8

@SvenRuppert

```
public interface CalculatorService {
  double calculate(double value);
}

// Default we want to Decorate
public static class CalculatorDefaultService implements CalculatorService {
  @Override
  public double calculate(final double value) {
    final double result = value + 100;
    System.out.println(this.getClass().getSimpleName() + " - in / out = " + value + " - " + result);
    return result;
  }
}
```

Decorator - pre Java8

@SvenRuppert

```java
public interface CalculatorService {
  double calculate(double value);
}

// Default we want to Decorate
public static class CalculatorDefaultService implements CalculatorService {
  @Override
  public double calculate(final double value) {
    final double result = value + 100;
    System.out.println(this.getClass().getSimpleName() + " - in / out = " + value + " - " + result);
    return result;
  }
}
```

```java
//not nice...
public static abstract class AbstractCalculatorService implements CalculatorService {
  private final CalculatorService delegator;

  public AbstractCalculatorService(final CalculatorService delegator) { this.delegator = delegator; }

  protected abstract double calculateImpl(final double value);

  public final double calculate(final double value) { return calculateImpl(delegator.calculate(value)); }
}
```

Decorator - pre Java8

@SvenRuppert

```java
public interface CalculatorService {
  double calculate(double value);
}

// Default we want to Decorate
public static class CalculatorDefaultService implements CalculatorService {
  @Override
  public double calculate(final double value) {
    final double result = value + 100;
    System.out.println(this.getClass().getSimpleName() + " - in / out = " + value + " - " + result);
    return result;
  }
}
```

```java
//not nice...
public static abstract class AbstractCalculatorService implements CalculatorService {
  private final CalculatorService delegator;

  public AbstractCalculatorService(final CalculatorService delegator) { this.delegator = delegator; }

  protected abstract double calculateImpl(final double value);

  public final double calculate(final double value) { return calculateImpl(delegator.calculate(value)); }
}
```

Decorator - pre Java8

@SvenRuppert

```java
public interface CalculatorService {
  double calculate(double value);
}

// Default we want to Decorate
public static class CalculatorDefaultService implements CalculatorService {
  @Override
  public double calculate(final double value) {
    final double result = value + 100;
    System.out.println(this.getClass().getSimpleName() + " - in / out = " + value + " - " + result);
    return result;
  }
}
```

```java
//not nice...
public static abstract class AbstractCalculatorService implements CalculatorService {
  private final CalculatorService delegator;

  public AbstractCalculatorService(final CalculatorService delegator) { this.delegator = delegator; }

  protected abstract double calculateImpl(final double value);

  public final double calculate(final double value) { return calculateImpl(delegator.calculate(value)); }
}
```

Decorator - pre Java8

@SvenRuppert

```java
public interface CalculatorService {
  double calculate(double value);
}

// Default we want to Decorate
public static class CalculatorDefaultService implements CalculatorService {
  @Override
  public double calculate(final double value) {
    final double result = value + 100;
    System.out.println(this.getClass().getSimpleName() + " - in / out = " + value + " - " + result);
    return result;
  }
}
```

```java
//not nice...
public static abstract class AbstractCalculatorService implements CalculatorService {
  private final CalculatorService delegator;

  public AbstractCalculatorService(final CalculatorService delegator) { this.delegator = delegator; }

  protected abstract double calculateImpl(final double value);

  public final double calculate(final double value) { return calculateImpl(delegator.calculate(value)); }
}
```

Decorator - pre Java8

@SvenRuppert

```java
public interface CalculatorService {
  double calculate(double value);
}

// Default we want to Decorate
public static class CalculatorDefaultService implements CalculatorService {
  @Override
  public double calculate(final double value) {
    final double result = value + 100;
    System.out.println(this.getClass().getSimpleName() + " - in / out = " + value + " - " + result);
    return result;
  }
}
```

```java
//not nice...
public static abstract class AbstractCalculatorService implements CalculatorService {
  private final CalculatorService delegator;

  public AbstractCalculatorService(final CalculatorService delegator) { this.delegator = delegator; }

  protected abstract double calculateImpl(final double value);

  public final double calculate(final double value) { return calculateImpl(delegator.calculate(value)); }
}
```

```java
public static class RoundUpService extends AbstractCalculatorService {
  public RoundUpService(final CalculatorService delegator) { super(delegator); }

  protected double calculateImpl(double value) {
    final double result = Math.ceil(value);
    System.out.println(this.getClass().getSimpleName() + " - in / out = " + value + " - " + result);
    return result;
  }
}
```

Decorator - pre Java8

@SvenRuppert

```java
public interface CalculatorService {
  double calculate(double value);
}

// Default we want to Decorate
public static class CalculatorDefaultService implements CalculatorService {
  @Override
  public double calculate(final double value) {
    final double result = value + 100;
    System.out.println(this.getClass().getSimpleName() + " - in / out = " + value + " - " + result);
    return result;
  }
}
```

```java
//not nice...
public static abstract class AbstractCalculatorService implements CalculatorService {
  private final CalculatorService delegator;

  public AbstractCalculatorService(final CalculatorService delegator) { this.delegator = delegator; }

  protected abstract double calculateImpl(final double value);

  public final double calculate(final double value) { return calculateImpl(delegator.calculate(value)); }
}
```

```java
public static class RoundUpService extends AbstractCalculatorService {
  public RoundUpService(final CalculatorService delegator) { super(delegator); }

  protected double calculateImpl(double value) {
    final double result = Math.ceil(value);
    System.out.println(this.getClass().getSimpleName() + " - in / out = " + value + " - " + result);
    return result;
  }
}
```

Decorator - pre Java8

@SvenRuppert

```java
public interface CalculatorService {
  double calculate(double value);
}

// Default we want to Decorate
public static class CalculatorDefaultService implements CalculatorService {
  @Override
  public double calculate(final double value) {
    final double result = value + 100;
    System.out.println(this.getClass().getSimpleName() + " - in / out = " + value + " - " + result);
    return result;
  }
}
```

```java
//not nice...
public static abstract class AbstractCalculatorService implements CalculatorService {
  private final CalculatorService delegator;

  public AbstractCalculatorService(final CalculatorService delegator) { this.delegator = delegator; }

  protected abstract double calculateImpl(final double value);

  public final double calculate(final double value) { return calculateImpl(delegator.calculate(value)); }
}
```

```java
public static class RoundUpService extends AbstractCalculatorService {
  public RoundUpService(final CalculatorService delegator) { super(delegator); }

  protected double calculateImpl(double value) {
    final double result = Math.ceil(value);
    System.out.println(this.getClass().getSimpleName() + " - in / out = " + value + " - " + result);
    return result;
  }
}
```

```java
public static void main(String[] args) {
  final CalculatorService calculatorService =
      new RoundUpService(
          new AddHalfService(
              new SubOneService(
                  new CalculatorDefaultService())));

  final double calculate = calculatorService.calculate( value: 10.3d);
  System.out.println("calculate = " + calculate);
```

Decorator - Java8

Decorator - Java8

@SvenRuppert

```
@FunctionalInterface
public interface CalculatorService {
  double calculate(double value);
}

@FunctionalInterface
public interface CalculatorDecoratorService extends CalculatorService {

  default CalculatorDecoratorService calculateBefore(CalculatorService before) {
    Objects.requireNonNull(before);
    return (double value) -> calculate(before.calculate(value));
  }
}
```

Decorator - Java8

@SvenRuppert

```
@FunctionalInterface
public interface CalculatorService {
  double calculate(double value);
}

@FunctionalInterface
public interface CalculatorDecoratorService extends CalculatorService {

  default CalculatorDecoratorService calculateBefore(CalculatorService before) {
    Objects.requireNonNull(before);
    return (double value) -> calculate(before.calculate(value));
  }
}
```

Decorator - Java8

@SvenRuppert

```java
@FunctionalInterface
public interface CalculatorService {
  double calculate(double value);
}

@FunctionalInterface
public interface CalculatorDecoratorService extends CalculatorService {

  default CalculatorDecoratorService calculateBefore(CalculatorService before) {
    Objects.requireNonNull(before);
    return (double value) -> calculate(before.calculate(value));
  }
}
```

```java
public static final CalculatorService CalculatorDefaultService = (value) -> {
  final double result = value + 100;
  System.out.println("CalculatorDefaultService - in / out = " + value + " - " + result);
  return result;
};
```

Decorator - Java8

@SvenRuppert

```java
@FunctionalInterface
public interface CalculatorService {
  double calculate(double value);
}

@FunctionalInterface
public interface CalculatorDecoratorService extends CalculatorService {

  default CalculatorDecoratorService calculateBefore(CalculatorService before) {
    Objects.requireNonNull(before);
    return (double value) -> calculate(before.calculate(value));
  }
}
```

```java
public static final CalculatorService CalculatorDefaultService = (value) -> {
  final double result = value + 100;
  System.out.println("CalculatorDefaultService - in / out = " + value + " - " + result);
  return result;
};
```

Decorator - Java8

@SvenRuppert

```java
@FunctionalInterface
public interface CalculatorService {
  double calculate(double value);
}

@FunctionalInterface
public interface CalculatorDecoratorService extends CalculatorService {

  default CalculatorDecoratorService calculateBefore(CalculatorService before) {
    Objects.requireNonNull(before);
    return (double value) -> calculate(before.calculate(value));
  }
}
```

```java
public static final CalculatorService CalculatorDefaultService = (value) -> {
  final double result = value + 100;
  System.out.println("CalculatorDefaultService - in / out = " + value + " - " + result);
  return result;
};
```

```java
public static final CalculatorDecoratorService RoundUpService = (value) -> {
  final double result = Math.ceil(value);
  System.out.println("RoundUpService - in / out = " + value + " - " + result);
  return result;
};
```

Decorator - Java8

@SvenRuppert

```java
@FunctionalInterface
public interface CalculatorService {
  double calculate(double value);
}

@FunctionalInterface
public interface CalculatorDecoratorService extends CalculatorService {

  default CalculatorDecoratorService calculateBefore(CalculatorService before) {
    Objects.requireNonNull(before);
    return (double value) -> calculate(before.calculate(value));
  }
}
```

```java
public static final CalculatorService CalculatorDefaultService = (value) -> {
  final double result = value + 100;
  System.out.println("CalculatorDefaultService - in / out = " + value + " - " + result);
  return result;
};
```

```java
public static final CalculatorDecoratorService RoundUpService = (value) -> {
  final double result = Math.ceil(value);
  System.out.println("RoundUpService - in / out = " + value + " - " + result);
  return result;
};
```

Decorator - Java8

@SvenRuppert

```java
@FunctionalInterface
public interface CalculatorService {
  double calculate(double value);
}

@FunctionalInterface
public interface CalculatorDecoratorService extends CalculatorService {

  default CalculatorDecoratorService calculateBefore(CalculatorService before) {
    Objects.requireNonNull(before);
    return (double value) -> calculate(before.calculate(value));
  }
}
```

```java
public static final CalculatorService CalculatorDefaultService = (value) -> {
  final double result = value + 100;
  System.out.println("CalculatorDefaultService - in / out = " + value + " - " + result);
  return result;
};
```

```java
public static final CalculatorDecoratorService RoundUpService = (value) -> {
  final double result = Math.ceil(value);
  System.out.println("RoundUpService - in / out = " + value + " - " + result);
  return result;
};
```

Decorator - Java8

@SvenRuppert

```java
@FunctionalInterface
public interface CalculatorService {
  double calculate(double value);
}

@FunctionalInterface
public interface CalculatorDecoratorService extends CalculatorService {

  default CalculatorDecoratorService calculateBefore(CalculatorService before) {
    Objects.requireNonNull(before);
    return (double value) -> calculate(before.calculate(value));
  }
}
```

Decorator - Java8

@SvenRuppert

```java
@FunctionalInterface
public interface CalculatorService {
  double calculate(double value);
}

@FunctionalInterface
public interface CalculatorDecoratorService extends CalculatorService {

  default CalculatorDecoratorService calculateBefore(CalculatorService before) {
    Objects.requireNonNull(before);
    return (double value) -> calculate(before.calculate(value));
  }
}
```

```java
final double calculate = RoundUpService
    .calculateBefore(AddHalfService)
    .calculateBefore(SubOneService)
    .calculateBefore(CalculatorDefaultService)
    .calculate( value: 10.3d);
```

Decorator - Java8

@SvenRuppert

```java
@FunctionalInterface
public interface CalculatorService {
  double calculate(double value);
}

@FunctionalInterface
public interface CalculatorDecoratorService extends CalculatorService {

  default CalculatorDecoratorService calculateBefore(CalculatorService before) {
    Objects.requireNonNull(before);
    return (double value) -> calculate(before.calculate(value));
  }
}
```

```java
final double calculate = RoundUpService
    .calculateBefore(AddHalfService)
    .calculateBefore(SubOneService)
    .calculateBefore(CalculatorDefaultService)
    .calculate( value: 10.3d);
```

```java
final double calculateShort = RoundUpService
    .calculateBefore(value -> value + 0.5d)
    .calculateBefore(value -> value - 1)
    .calculateBefore(CalculatorDefaultService)
    .calculate( value: 10.3d);
```

Decorator - Java8 - modify orig Interface

@SvenRuppert

```
@FunctionalInterface
public interface CalculatorService {
  double calculate(double value);
}

@FunctionalInterface
public interface CalculatorDecoratorService extends CalculatorService {

  default CalculatorDecoratorService calculateBefore(CalculatorService before) {
    Objects.requireNonNull(before);
    return (double value) -> calculate(before.calculate(value));
  }
}
```

Decorator - Java8 - modify orig Interface

@SvenRuppert

```
@FunctionalInterface
public interface CalculatorService {
  double calculate(double value);
}

@FunctionalInterface
public interface CalculatorDecoratorService extends CalculatorService {

  default CalculatorDecoratorService calculateBefore(CalculatorService before) {
    Objects.requireNonNull(before);
    return (double value) -> calculate(before.calculate(value));
  }
}
```

```
@FunctionalInterface
public interface CalculatorService {

  double calculate(double value);

  default CalculatorService andThen(CalculatorService after) {
    Objects.requireNonNull(after);
    return (double value) -> after.calculate(calculate(value));
  }
}
```

Decorator - Java8 - modify orig Interface

@SvenRuppert

```
@FunctionalInterface
public interface CalculatorService {
  double calculate(double value);
}

@FunctionalInterface
public interface CalculatorDecoratorService extends CalculatorService {

  default CalculatorDecoratorService calculateBefore(CalculatorService before) {
    Objects.requireNonNull(before);
    return (double value) -> calculate(before.calculate(value));
  }
}
```

```
@FunctionalInterface
public interface CalculatorService {

  double calculate(double value);

  default CalculatorService andThen(CalculatorService after) {
    Objects.requireNonNull(after);
    return (double value) -> after.calculate(calculate(value));
  }
}
```

Decorator - Java8 - modify orig Interface

@SvenRuppert

```
@FunctionalInterface
public interface CalculatorService {
  double calculate(double value);
}

@FunctionalInterface
public interface CalculatorDecoratorService extends CalculatorService {

  default CalculatorDecoratorService calculateBefore(CalculatorService before) {
    Objects.requireNonNull(before);
    return (double value) -> calculate(before.calculate(value));
  }
}
```

```
@FunctionalInterface
public interface CalculatorService {

  double calculate(double value);

  default CalculatorService andThen(CalculatorService after) {
    Objects.requireNonNull(after);
    return (double value) -> after.calculate(calculate(value));
  }
}
```

Decorator - Java8 - modify orig Interface

@SvenRuppert

```java
@FunctionalInterface
public interface CalculatorService {
  double calculate(double value);
}

@FunctionalInterface
public interface CalculatorDecoratorService extends CalculatorService {

  default CalculatorDecoratorService calculateBefore(CalculatorService before) {
    Objects.requireNonNull(before);
    return (double value) -> calculate(before.calculate(value));
  }
}
```

```java
@FunctionalInterface
public interface CalculatorService {

  double calculate(double value);

  default CalculatorService andThen(CalculatorService after) {
    Objects.requireNonNull(after);
    return (double value) -> after.calculate(calculate(value));
  }
}
```

Decorator - Java8 - DoubleUnaryOperator

@SvenRuppert

Decorator - Java8 - DoubleUnaryOperator

@SvenRuppert

```
//  public interface CalculatorService extends DoubleUnaryOperator {
//    double applyAsDouble(double value);
//  }
```

Decorator - Java8 - DoubleUnaryOperator

@SvenRuppert

```
//  public interface CalculatorService extends DoubleUnaryOperator {
//    double applyAsDouble(double value);
//  }
```

```
public static final DoubleUnaryOperator CalculatorDefaultService = (value) -> {
  final double result = value + 100;
  System.out.println("CalculatorDefaultService - in / out = " + value + " - " + result);
  return result;
};
```

Decorator - Java8 - DoubleUnaryOperator

@SvenRuppert

```
//  public interface CalculatorService extends DoubleUnaryOperator {
//    double applyAsDouble(double value);
//  }
```

```
public static final DoubleUnaryOperator CalculatorDefaultService = (value) -> {
  final double result = value + 100;
  System.out.println("CalculatorDefaultService - in / out = " + value + " - " + result);
  return result;
};
```

```
final double calculate = CalculatorDefaultService
    .andThen(SubOneService)
    .andThen(AddHalfService)
    .andThen(RoundUpService)
    .applyAsDouble( operand: 10.3d);
```

Decorator - Java8 - DoubleUnaryOperator

@SvenRuppert

```
//  public interface CalculatorService extends DoubleUnaryOperator {
//    double applyAsDouble(double value);
//  }
```

```
public static final DoubleUnaryOperator CalculatorDefaultService = (value) -> {
  final double result = value + 100;
  System.out.println("CalculatorDefaultService - in / out = " + value + " - " + result);
  return result;
};
```

```
final double calculate = CalculatorDefaultService
    .andThen(SubOneService)
    .andThen(AddHalfService)
    .andThen(RoundUpService)
    .applyAsDouble( operand: 10.3d);
```

```
final double calculateShort = CalculatorDefaultService
    .andThen(value -> value - 1)
    .andThen(value -> value + 0.5d)
    .andThen(Math::ceil)
    .applyAsDouble( operand: 10.3d);
```

Interpreter - pre Java8

Interpreter - pre Java8

@SvenRuppert

```
interface Expression {
  int interpret();
}

public static abstract class Operator implements Expression {
  protected final Expression leftExpression;
  protected final Expression rightExpression;

  public Operator(final Expression leftExpression, final Expression rightExpression) {
    this.leftExpression = leftExpression;
    this.rightExpression = rightExpression;
  }
}
```

Interpreter - pre Java8

@SvenRuppert

```
interface Expression {
  int interpret();
}

public static abstract class Operator implements Expression {
  protected final Expression leftExpression;
  protected final Expression rightExpression;

  public Operator(final Expression leftExpression, final Expression rightExpression) {
    this.leftExpression = leftExpression;
    this.rightExpression = rightExpression;
  }
}
```

```
public static class Add extends Operator {
  public Add(final Expression leftExpression, final Expression rightExpression) {
    super(leftExpression, rightExpression);
  }

  @Override
  public int interpret() {
    return leftExpression.interpret() + rightExpression.interpret();
  }
}
```

Interpreter - pre Java8

Interpreter - pre Java8

@SvenRuppert

```
public static class Number implements Expression {
  private final int n;

  public Number(int n) { this.n = n; }

  @Override
  public int interpret() { return n; }
}
```

Interpreter - pre Java8

@SvenRuppert

```
public static class Number implements Expression {
  private final int n;

  public Number(int n) { this.n = n; }

  @Override
  public int interpret() { return n; }
}
```

```
public static Expression getOperator(String s, Expression left, Expression right) {
  return
      (s.equals("+")) ? new Add(left, right) :
      (s.equals("-")) ? new Subtract(left, right) :
      (s.equals("*")) ? new Product(left, right) :
                        null; // not nice ;-)
}
```

Interpreter - pre Java8

Interpreter - pre Java8

@SvenRuppert

```
public static boolean isOperator(String s) {
  return s.equals("+") || s.equals("-") || s.equals("*");
}
```

Interpreter - pre Java8

@SvenRuppert

```java
public static boolean isOperator(String s) {
  return s.equals("+") || s.equals("-") || s.equals("*");
}
```

```java
public static int evaluate(String expression) {
  final Stack<Expression> stack = new Stack<>();
  for (final String s : expression.split( regex: " ")) {
    if (isOperator(s)) {
      Expression right = stack.pop();
      Expression left = stack.pop();
      stack.push(getOperator(s, left, right));
    } else {
      Expression i = new Number(Integer.parseInt(s));
      stack.push(i);
    }
  }
  return stack.pop().interpret();
}
```

Interpreter - pre Java8

@SvenRuppert

```
public static boolean isOperator(String s) {
  return s.equals("+") || s.equals("-") || s.equals("*");
}
```

```
public static int evaluate(String expression) {
  final Stack<Expression> stack = new Stack<>();
  for (final String s : expression.split( regex: " ")) {
    if (isOperator(s)) {
      Expression right = stack.pop();
      Expression left = stack.pop();
      stack.push(getOperator(s, left, right));
    } else {
      Expression i = new Number(Integer.parseInt(s));
      stack.push(i);
    }
  }
  return stack.pop().interpret();
}
```

```
public static void main(String[] args) {
  String expression = "7 3 - 2 1 + *";
  System.out.println(evaluate(expression));
}
```

Interpreter - Java8 - Java9

Interpreter - Java8 - Java9

@SvenRuppert

```
private static final Map<String, IntBinaryOperator> OPERATOR_MAP = new HashMap<>();

static {
  OPERATOR_MAP.put("+", (a, b) -> a + b);
  OPERATOR_MAP.put("*", (a, b) -> a * b);
  OPERATOR_MAP.put("-", (a, b) -> a - b);
}
```

Interpreter - Java8 - Java9

@SvenRuppert

```java
private static final Map<String, IntBinaryOperator> OPERATOR_MAP = new HashMap<>();

static {
  OPERATOR_MAP.put("+", (a, b) -> a + b);
  OPERATOR_MAP.put("*", (a, b) -> a * b);
  OPERATOR_MAP.put("-", (a, b) -> a - b);
}
```

```java
public static int evaluate(String expression) {
  Stack<Integer> stack = new Stack<>();
  for (final String s : expression.split( regex: " ")) {
    IntBinaryOperator op = OPERATOR_MAP.get( s );
    if (op != null) {
      int right = stack.pop();
      int left = stack.pop();
      stack.push(op.applyAsInt( left, right ));
    } else {
      stack.push(Integer.parseInt(s));
    }
  }
  return stack.pop();
}
```

Interpreter - Java8 - Java9

@SvenRuppert

show code inside the IDE

Main Idea - DataStructure to Function

Interpreter - Java8 - Java9

@SvenRuppert

Main Idea - DataStructure to Function

```
private static final Map<String, IntBinaryOperator> OPERATOR_MAP = new HashMap<>();

static {
  OPERATOR_MAP.put("+", (a, b) -> a + b);
  OPERATOR_MAP.put("*", (a, b) -> a * b);
  OPERATOR_MAP.put("-", (a, b) -> a - b);
}
```

Interpreter - Java8 - Java9

@SvenRuppert

Main Idea - DataStructure to Function

```
private static final Map<String, IntBinaryOperator> OPERATOR_MAP = new HashMap<>();

static {
  OPERATOR_MAP.put("+", (a, b) -> a + b);
  OPERATOR_MAP.put("*", (a, b) -> a * b);
  OPERATOR_MAP.put("-", (a, b) -> a - b);
}
```

```
private static final Function<String, Optional<IntBinaryOperator>> str2OpFNC = (str) ->
    (isNull(str))      ? empty() :
    (str.equals("+")) ? of((a, b) -> a + b) :
    (str.equals("*")) ? of((a, b) -> a * b) :
    (str.equals("-")) ? of((a, b) -> a - b) :
                        empty();
```

Java8 - Streams

Java8 - Streams

Streams are

Java8 - Streams

Streams are

.. not a data structure

Java8 - Streams

Streams are

.. not a data structure

..using Lambdas

Java8 - Streams

Streams are

.. not a data structure

..using Lambdas

..no access to dedicated elements

Java8 - Streams

Streams are

.. not a data structure

..using Lambdas

..no access to dedicated elements

.. are lazy

Java8 - Streams

Streams are

.. not a data structure

..using Lambdas

..no access to dedicated elements

.. are lazy

.. are parallel if needed

Java8 - Streams

Streams are

.. not a data structure

..using Lambdas

..no access to dedicated elements

.. are lazy

.. are parallel if needed

.. are unbounded

Java8 - Streams - data in

How to create Streams and how to get data in

Java8 - Streams - data in

How to create Streams and how to get data in

Stream.**of(val1,val2,val3…)**

Java8 - Streams - data in

How to create Streams and how to get data in

Stream.**of(val1,val2,val3…)**

Stream.**of(array)**

Java8 - Streams - data in

How to create Streams and how to get data in

Stream.**of(val1,val2,val3…)**

Stream.**of(array)**

list.**stream()**

Java8 - Streams - data in

How to create Streams and how to get data in

Stream.**of(val1,val2,val3…)**

Stream.**of(array)**

list.**stream()**

Stream.**<Pair>builder().add(new Pair()).build()**

Java8 - Streams - data in

@SvenRuppert

How to create Streams and how to get data in

Stream.**of(val1,val2,val3...)** Stream.**of(array)**

list.**stream()** Stream.**<Pair>builder().add(new Pair()).build()**

```
Stream.generate(() -> {
    final Pair p = new Pair();
    p.id = random.nextInt(100);
    p.value = "Value + " + p.id;
    return p;
  })
```

Java8 - Streams - data in

@SvenRuppert

How to create Streams and how to get data in

```
Stream.generate(new Supplier<String>() {
    private int counter = 0;
    @Override
    public String get() {
        if (counter < 10) {
            final String result = counter + "";
            counter++;
            return result;
        }
        return null;
    }
})
        .limit( maxSize: 20)
        .forEach(System.out::println);
```

Java8 - Streams - data out

How to create Streams and how to get data out

Java8 - Streams - data out

How to create Streams and how to get data out

stream.**toArray()**;

Java8 - Streams - data out

How to create Streams and how to get data out

stream.**toArray()**;

stream.collect(**Collectors.toList()**);

Java8 - Streams - data out

@SvenRuppert

How to create Streams and how to get data out

```
stream.toArray();

stream.collect(Collectors.toList());

stream.collect(Collectors.toSet());
```

Java8 - Streams - data out

@SvenRuppert

How to create Streams and how to get data out

```
stream.toArray();

stream.collect(Collectors.toList());

stream.collect(Collectors.toSet());

stream.collect(Collectors.groupingBy(..))
```

Java8 - Streams - data out

@SvenRuppert

How to create Streams and how to get data out

```
final List<Pair> generateDemoValues = generateDemoValues();

final Pair[] toArray =
    generateDemoValues.stream()
    .toArray(Pair[]::new);

final Map<Integer,List<Pair>> collectedToMap =
    generateDemoValues.stream()
    .collect(Collectors.groupingBy(Pair::getId));
```

Java8 - Streams - data out

How to create Streams and how to get data in/out

- easy to combine with legacy code
  - start with transformation of loops/if blocks
    - reduce code

Java8 - Streams - Core Functions

Java8 - Streams - Core Functions

forEach

Java8 - Streams - Core Functions

forEach

limit / skip

Java8 - Streams - Core Functions

forEach

limit / skip

distinct

Java8 - Streams - Core Functions

@SvenRuppert

forEach

limit / skip

distinct

map

Java8 - Streams - Core Functions

forEach

limit / skip

distinct

map

reduce

Java8 - Streams - Core Functions

@SvenRuppert

forEach

limit / skip

distinct

map

reduce

min / max

Java8 - Streams - Core Functions

@SvenRuppert

forEach

limit / skip

distinct

map

reduce

findFirst

min / max

Java8 - Streams - Core Functions

@SvenRuppert

forEach

limit / skip

distinct

map

reduce

findFirst

filter

min / max

Java8 - Streams - Core Functions

@SvenRuppert

forEach

limit / skip

distinct

map

reduce

findFirst

filter

min / max

allMatch / anyMatch / noneMatch

Java8 - Streams - Core Functions

@SvenRuppert

forEach

limit / skip

distinct

map

reduce

findFirst

filter

min / max

allMatch / anyMatch / noneMatch

parallel / sequenziell

Java8 - Streams - Core Functions

Java8 - Streams - Core Functions

@SvenRuppert

```
final List<Pair> generateDemoValues = new PairListGenerator(){}.generateDemoValues();

//pre JDK8
for (final Pair generateDemoValue : generateDemoValues) {
    System.out.println(generateDemoValue);
}

//long version
generateDemoValues.stream().forEach(v -> System.out.println(v) );

//short version
generateDemoValues.stream().forEach(System.out::println);
generateDemoValues.parallelStream().forEach(System.out::println);


generateDemoValues.stream().forEachOrdered(System.out::println);
generateDemoValues.parallelStream().forEachOrdered(System.out::println);


final Consumer<? super Pair> consumer = System.out::println;
generateDemoValues.stream().forEachOrdered(consumer);
generateDemoValues.parallelStream().forEachOrdered(consumer);
```

Java8 - Streams - Core Functions

Java8 - Streams - Core Functions

@SvenRuppert

```
final Stream<String> stringStream = getDemoElementStream()
    .map(v -> v.getValue());
final Stream<String> stringStreamShort = getDemoElementStream()
    .map(DemoElement::getValue);
```

Java8 - Streams - Core Functions

@SvenRuppert

```
final Stream<String> stringStream = getDemoElementStream()
    .map(v -> v.getValue());
final Stream<String> stringStreamShort = getDemoElementStream()
    .map(DemoElement::getValue);
```

```
final Stream<String> stringStreamFull = getDemoElementStream()
    .map(new Function<DemoElement, String>() {
      @Override
      public String apply(final DemoElement demoElement) {
        return demoElement.getValue();
      }
    });
```

Java8 - Streams - Core Functions

Java8 - Streams - Core Functions

@SvenRuppert

```
private static final List<Pair> generateDemoValues = new PairListGenerator() {}.generateDemoValues();
```

Java8 - Streams - Core Functions

@SvenRuppert

```
private static final List<Pair> generateDemoValues = new PairListGenerator() {}.generateDemoValues();
```

```
private static final Function<Pair, DemoElement> pairDemoElementFunction = v -> {
  final String value = v.getValue();
  final DemoElement d = new DemoElement();
  d.setDatum(new Date());
  d.setValue(Base64.getEncoder().encodeToString(value.getBytes()));
  return d;
};
```

Java8 - Streams - Core Functions

@SvenRuppert

```
private static final List<Pair> generateDemoValues = new PairListGenerator() {}.generateDemoValues();
```

```
private static final Function<Pair, DemoElement> pairDemoElementFunction = v -> {
  final String value = v.getValue();
  final DemoElement d = new DemoElement();
  d.setDatum(new Date());
  d.setValue(Base64.getEncoder().encodeToString(value.getBytes()));
  return d;
};
```

```
private static Stream<DemoElement> getDemoElementStream() {
  return generateDemoValues
      .stream()
      .map(pairDemoElementFunction);
}
```

Java8 - Streams - Core Functions

Java8 - Streams - Core Functions

@SvenRuppert

```
private static final List<Pair> generateDemoValues = new PairListGenerator() {
}.generateDemoValues();

public static void main(String[] args) {
  generateDemoValues
      .stream()
      .map(Pair::getValue)
      .mapToInt(e1 -> (int) e1.charAt(9))
      .mapToObj(i -> {
        class Holder {
          int h;
        }
        final Holder holder = new Holder();
        holder.h = i;
        return holder;
      })
      .map(h -> h.h)
      .forEach(System.out::println);

}
```

Java8 - Streams - Core Functions

Java8 - Streams - Core Functions

@SvenRuppert

```
generateDemoValues.stream()
    .filter(v -> v.getId() % 2 == 0)
    .filter(new Predicate<Pair>() {
      @Override
      public boolean test(final Pair v) {
        return v.getValue().contains("5");
      }
    })
.forEach(System.out::println);
```

Java8 - Streams - Core Functions

@SvenRuppert

```
generateDemoValues.stream()
    .filter(v -> v.getId() % 2 == 0)
    .filter(new Predicate<Pair>() {
      @Override
      public boolean test(final Pair v) {
        return v.getValue().contains("5");
      }
    })
.forEach(System.out::println);
```

```
generateDemoValues.stream()
    .filter(v -> v.getId() % 2 == 0)
    .filter(v->v.getValue().contains("5"))
.forEach(System.out::println);
```

Java8 - Streams - Core Functions

Java8 - Streams - Core Functions

@SvenRuppert

```
final List<String> demoValues = Arrays.asList("A", "B", "C", "D", "E");

System.out.println(demoValues
    .stream()
    .reduce(String::concat)); //Optional[ABCDE]

System.out.println(demoValues
    .parallelStream()
    .reduce(String::concat));
```

Java8 - Streams - Core Functions

Java8 - Streams - Core Functions

@SvenRuppert

```
System.out.println(demoValues
    .stream()
    .reduce( identity: "", String::concat));
```

Java8 - Streams - Core Functions

@SvenRuppert

```
System.out.println(demoValues
        .stream()
        .reduce( identity: "", String::concat));
```

```
//result is X_ABCDE
System.out.println(demoValues
        .stream()
        .reduce( identity: "X_", String::concat));
```

Java8 - Streams - Core Functions

@SvenRuppert

```
//result is X_ABCDE
System.out.println(demoValues
    .stream()
    .reduce( identity: "X_", String::concat));
```

Java8 - Streams - Core Functions

@SvenRuppert

```
//result is X_ABCDE
System.out.println(demoValues
    .stream()
    .reduce( identity: "X_", String::concat));
```

```
//result is X_AX_BX_CX_DX_E
System.out.println(demoValues
    .parallelStream()
    .reduce( identity: "X_", String::concat));
```

Java8 - Streams - Core Functions

Java8 - Streams - Core Functions

@SvenRuppert

```
final List<Integer> demoIntValues = Arrays.asList(1, 2, 3, 4, 5);

final Integer maxInteger = demoIntValues
    .parallelStream()
    .reduce(Integer.MIN_VALUE, Integer::max);
System.out.println("maxInteger = " + maxInteger);
```

Java8 - Streams - Core Functions

Java8 - Streams - Core Functions

@SvenRuppert

```
final List<Integer> demoValues
        = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

//limit the input
System.out.println(demoValues
        .stream().limit( maxSize: 4)
        .collect(Collectors.toList()));
```

Java8 - Streams - Core Functions

@SvenRuppert

```
final List<Integer> demoValues
        = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

//limit the input
System.out.println(demoValues
        .stream().limit(maxSize: 4)
        .collect(Collectors.toList()));
```

```
//limit the result
System.out.println(demoValues
        .stream().filter((v) -> v > 4)
        .limit(maxSize: 4)
        .collect(Collectors.toList()));
```

Java8 - Streams - Core Functions

Java8 - Streams - Core Functions

@SvenRuppert

```
System.out.println(demoValues
        .stream().skip( n: 4)
        .collect(Collectors.toList()));
```

Java8 - Streams - Core Functions

Java8 - Streams - Core Functions

@SvenRuppert

```
System.out.println(demoValues
        .stream().sorted((v1, v2) -> Integer.compare(v2, v1))
        .collect(Collectors.toList()));
```

Java8 - Streams - Core Functions

@SvenRuppert

```
System.out.println(demoValues
        .stream().sorted((v1, v2) -> Integer.compare(v2, v1))
        .collect(Collectors.toList()));
```

```
System.out.println(demoValues
    .stream().unordered()
    .collect(Collectors.toList()));
```

Java8 - Streams - Core Functions

Java8 - Streams - Core Functions

@SvenRuppert

```
// example result:
// [77, 79, 81, 95, 43, 10, 53, 48, 74, 68, 60, 86, 83, 24, 57, 28, 8,
//  85, 70, 66, 20, 14, 97, 73, 22, 36, 40, 39, 32, 19, 41, 67, 25, 88]
final Random random = new Random();
System.out.println(
        Stream.generate(() -> random.nextInt( bound: 100))
                .limit( maxSize: 40)
                .distinct()
                .collect(Collectors.toList())
);
```

Java8 - Streams - Core Functions

@SvenRuppert

```
// example result:
// [77, 79, 81, 95, 43, 10, 53, 48, 74, 68, 60, 86, 83, 24, 57, 28, 8,
//  85, 70, 66, 20, 14, 97, 73, 22, 36, 40, 39, 32, 19, 41, 67, 25, 88]
final Random random = new Random();
System.out.println(
        Stream.generate(() -> random.nextInt( bound: 100))
                .limit( maxSize: 40)
                .distinct()
                .collect(Collectors.toList())
);
```

```
// If the stream is SORTED then it should also be ORDERED so the following will also
// preserve the sort order
TerminalOp<T, LinkedHashSet<T>> reduceOp
        = ReduceOps.<~>makeRef(LinkedHashSet::new, LinkedHashSet::add,
                                                    LinkedHashSet::addAll);
return Nodes.node(reduceOp.evaluateParallel(helper, spliterator));
```

Java8 - Streams - Core Functions

Java8 - Streams - Core Functions

@SvenRuppert

```
final List<Integer> demoValues
    = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

//find the maximum
System.out.println(demoValues
    .stream().max(Integer::compareTo)
);
//find the BUG ;-)
System.out.println(demoValues
    .stream().min((v1, v2) -> Integer.compare(v2, v1))
);
```

Java8 - Streams - Core Functions

Java8 - Streams - Core Functions

@SvenRuppert

```
public static final List<Integer> demoValues
        = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

private static Stream<Integer> getStream() { return demoValues.stream(); }

static final Function<Integer, Integer> f = (e) -> {
    System.out.println("e = " + e);
    return e;
};
```

Java8 - Streams - Core Functions

@SvenRuppert

```
public static final List<Integer> demoValues
        = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

private static Stream<Integer> getStream() { return demoValues.stream(); }

static final Function<Integer, Integer> f = (e) -> {
    System.out.println("e = " + e);
    return e;
};

// true, some are matching
System.out.println("anyMatch " + getStream().map(f).anyMatch((v) -> v % 2 == 0));
```

Java8 - Streams - Core Functions

@SvenRuppert

```
public static final List<Integer> demoValues
        = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

private static Stream<Integer> getStream() { return demoValues.stream(); }

static final Function<Integer, Integer> f = (e) -> {
    System.out.println("e = " + e);
    return e;
};
```

```
// true, some are matching
System.out.println("anyMatch " + getStream().map(f).anyMatch((v) -> v % 2 == 0));
```

```
//false, not all are matching
System.out.println("allMatch " + getStream().map(f)
        .allMatch((v) -> v % 2 == 0));
```

Java8 - Streams - Core Functions

@SvenRuppert

```
public static final List<Integer> demoValues
        = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

private static Stream<Integer> getStream() { return demoValues.stream(); }

static final Function<Integer, Integer> f = (e) -> {
    System.out.println("e = " + e);
    return e;
};
```

```
// true, some are matching
System.out.println("anyMatch " + getStream().map(f).anyMatch((v) -> v % 2 == 0));
```

```
//false, not all are matching
System.out.println("allMatch " + getStream().map(f)
        .allMatch((v) -> v % 2 == 0));
```

```
//false, not all are NOT matching
System.out.println("noneMatch " + getStream().map(f)
        .noneMatch((v) -> v % 2 == 0));
```

Java8 - Streams - Transformations

@SvenRuppert

```
public interface Worker {

    public static final int ANZAHL_KURVEN = 200;
    public static final int ANZAHL_MESSWERTE = 10;
    public static final int MAX_GENERATED_INT = 100;

    public abstract List<List<Integer>> generateDemoValueMatrix();

    public abstract List<List<Double>> generateInterpolatedValues(List<List<Integer>> baseValues);
}
```

Java8 - Streams - Seriell

@SvenRuppert

```
public class WorkerSerial implements Worker {

    @Override
    public List<List<Double>> generateInterpolatedValues(List<List<Integer>> baseValues) {
        final WorkLoadGenerator generator = new WorkLoadGenerator();
        final List<List<Double>> result = new ArrayList<>();
        for (final List<Integer> valueList : baseValues) {
            final List<Double> doubleList = generator.generate(valueList);
            result.add(doubleList);
        }
        return result;
    }


    private DemoValueGenerator valueGenerator = new DemoValueGenerator(){};

    public List<List<Integer>> generateDemoValueMatrix() {
        final List<List<Integer>> result = new ArrayList<>();
        for (int i = 0; i < ANZAHL_KURVEN; i++) {
            final List<Integer> demoValuesForY = valueGenerator.generateDemoValuesForY();
            result.add(demoValuesForY);
        }
        return result;
    }
}
```

Java8 - Streams - ExecutorService

@SvenRuppert

Java8 - Streams - ExecutorService

@SvenRuppert

```
public class WorkerParallelExecutorService implements Worker {

  private final ExecutorService executorService;

  public WorkerParallelExecutorService(ExecutorService executorService) {
    this.executorService = executorService;
  }
```

Java8 - Streams - ExecutorService

@SvenRuppert

```
@Override
public List<List<Integer>> generateDemoValueMatrix() {

  final Collection<? extends Callable<List<Integer>>> taskList
      = Stream.generate(
          () -> (Callable<List<Integer>>) () -> new DemoValueGenerator() {}.generateDemoValuesForY())
      .limit(ANZAHL_KURVEN)
      .collect(Collectors.toList());

  final List<List<Integer>> result = new ArrayList<>();
  try {
    final List<Future<List<Integer>>> futureList = executorService.invokeAll(taskList);
    for (final Future<List<Integer>> future : futureList) {
      final List<Integer> valueList = future.get();
      result.add(valueList);
    }
  } catch (InterruptedException | ExecutionException e) {
    e.printStackTrace();
  }
  return result;
}
```

Java8 - Streams - ExecutorService

@SvenRuppert

```
@Override
public List<List<Double>> generateInterpolatedValues(List<List<Integer>> baseValues) {

  final List<Callable<List<Double>>> taskList = baseValues.stream()
      .map(baseLine -> (Callable<List<Double>>) () -> new WorkLoadGenerator().generate(baseLine))
      .collect(Collectors.toList());

  final List<List<Double>> result = new ArrayList<>();
  try {
    final List<Future<List<Double>>> futureList = executorService.invokeAll(taskList);
    for (final Future<List<Double>> future : futureList) {
      final List<Double> valueList = future.get();
      result.add(valueList);
    }
  } catch (InterruptedException | ExecutionException e) {
    e.printStackTrace();
  }
  return result;
}
```

Java8 - Streams - Streams - remember

@SvenRuppert

```java
public class WorkerSerial implements Worker {

    @Override
    public List<List<Double>> generateInterpolatedValues(List<List<Integer>> baseValues) {
        final WorkLoadGenerator generator = new WorkLoadGenerator();
        final List<List<Double>> result = new ArrayList<>();
        for (final List<Integer> valueList : baseValues) {
            final List<Double> doubleList = generator.generate(valueList);
            result.add(doubleList);
        }
        return result;
    }


    private DemoValueGenerator valueGenerator = new DemoValueGenerator(){};

    public List<List<Integer>> generateDemoValueMatrix() {
        final List<List<Integer>> result = new ArrayList<>();
        for (int i = 0; i < ANZAHL_KURVEN; i++) {
            final List<Integer> demoValuesForY = valueGenerator.generateDemoValuesForY();
            result.add(demoValuesForY);
        }
        return result;
    }
}
```

Java8 - Streams - Streams

@SvenRuppert

```
public List<List<Double>> generateInterpolatedValuesStep01(List<List<Integer>> baseValues) {
  final WorkLoadGenerator generator = new WorkLoadGenerator();
  final List<List<Double>> result = new ArrayList<>();
  for (final List<Integer> valueList : baseValues) {
    final List<Double> doubleList = generator.generate(valueList);
    result.add(doubleList);
  }
  return result;
}

public List<List<Integer>> generateDemoValueMatrixStep01() {
  return Stream.generate(new Supplier<List<Integer>>() {
    @Override
    public List<Integer> get() {
      return valueGenerator.generateDemoValuesForY();
    }
  })
      .limit(ANZAHL_KURVEN)
      .collect(Collectors.toList());
}
```

Java8 - Streams - Streams

@SvenRuppert

```
public List<List<Double>> generateInterpolatedValuesStep02(List<List<Integer>> baseValues) {
  final WorkLoadGenerator generator = new WorkLoadGenerator();
  final List<List<Double>> result = new ArrayList<>();
  for (final List<Integer> valueList : baseValues) {
    final List<Double> doubleList = generator.generate(valueList);
    result.add(doubleList);
  }
  return result;
}

public List<List<Integer>> generateDemoValueMatrixStep02() {
  return Stream.generate(() -> valueGenerator.generateDemoValuesForY())
      .limit(ANZAHL_KURVEN)
      .collect(Collectors.toList());
}
```

Java8 - Streams - Streams

@SvenRuppert

```
public List<List<Double>> generateInterpolatedValuesStep03(List<List<Integer>> baseValues) {
  final WorkLoadGenerator generator = new WorkLoadGenerator();
  return baseValues.stream()
      .map(new Function<List<Integer>, List<Double>>() {
        @Override
        public List<Double> apply(final List<Integer> valueList) {
          return generator.generate(valueList);
        }
      })
      .collect(Collectors.toList());

}

public List<List<Integer>> generateDemoValueMatrixStep03() {
  return Stream.generate(() -> valueGenerator.generateDemoValuesForY())
      .limit(ANZAHL_KURVEN)
      .collect(Collectors.toList());
}
```

Java8 - Streams - Streams

@SvenRuppert

```
public List<List<Double>> generateInterpolatedValuesStep04(List<List<Integer>> baseValues) {
  final WorkLoadGenerator generator = new WorkLoadGenerator();
  return baseValues.stream()
      .map(valueList -> generator.generate(valueList))
      .collect(Collectors.toList());

}

public List<List<Integer>> generateDemoValueMatrixStep04() {
  return Stream.generate(() -> valueGenerator.generateDemoValuesForY())
      .limit(ANZAHL_KURVEN)
      .collect(Collectors.toList());
}
```

Java8 - Streams - Streams

@SvenRuppert

```
public List<List<Double>> generateInterpolatedValuesStep05(List<List<Integer>> baseValues) {
  final WorkLoadGenerator generator = new WorkLoadGenerator();
  return baseValues.stream()
      .map(generator::generate)
      .collect(Collectors.toList());

}

public List<List<Integer>> generateDemoValueMatrixStep05() {
  return Stream.generate(() -> valueGenerator.generateDemoValuesForY())
      .limit(ANZAHL_KURVEN)
      .collect(Collectors.toList());
}
```

Java8 - Streams - Streams

@SvenRuppert

```
public List<List<Double>> generateInterpolatedValuesStep06() {
  final WorkLoadGenerator generator = new WorkLoadGenerator();
  return Stream.generate(() -> valueGenerator.generateDemoValuesForY())
      .limit(ANZAHL_KURVEN)
      .collect(Collectors.toList())
      .stream()
      .map(generator::generate)
      .collect(Collectors.toList());

}
```

Java8 - Streams - Streams

@SvenRuppert

```
public List<List<Double>> generateInterpolatedValuesStep07() {
  final WorkLoadGenerator generator = new WorkLoadGenerator();
  return Stream.generate(this::generateDemoValuesForY)
      .limit(ANZAHL_KURVEN)
      .collect(Collectors.toList())
      .stream()
      .map(generator::generate)
      .collect(Collectors.toList());
}

public List<Integer> generateDemoValuesForY() {
  final Random random = new Random();
  return Stream
      .generate(() -> random.nextInt(Worker.MAX_GENERATED_INT))
      .limit(Worker.ANZAHL_MESSWERTE)
      .collect(Collectors.toList());
}
```

Java8 - Streams - Streams

@SvenRuppert

```
public List<List<Double>> generateInterpolatedValuesStep08() {
  final WorkLoadGenerator generator = new WorkLoadGenerator();
  return Stream.generate(this::generateDemoValuesForY08)
      .limit(ANZAHL_KURVEN)
      .collect(Collectors.toList())
      .stream()
      .map(generator::generate)
      .collect(Collectors.toList());
}

public List<Integer> generateDemoValuesForY08() {
  return new Random()
      .ints( randomNumberOrigin: 0, Worker.MAX_GENERATED_INT)
      .limit(Worker.ANZAHL_MESSWERTE)
      .boxed()
      .collect(Collectors.toList());
}
```

Java8 - Streams - Streams

@SvenRuppert

```
public List<List<Double>> generateInterpolatedValuesStep09() {
  final WorkLoadGenerator generator = new WorkLoadGenerator();
  return Stream.generate(
      () -> new Random()
          .ints( randomNumberOrigin: 0, Worker.MAX_GENERATED_INT)
          .limit(Worker.ANZAHL_MESSWERTE)
          .boxed()
          .collect(Collectors.toList()))
      .limit(ANZAHL_KURVEN)
      .collect(Collectors.toList())
      .stream()
      .map(generator::generate)
      .collect(Collectors.toList());
}
```

Java8 - Streams - Streams

@SvenRuppert

```
public List<List<Double>> generateInterpolatedValuesStep10() {
  return Stream.generate(
      () -> new Random()
          .ints( randomNumberOrigin: 0, Worker.MAX_GENERATED_INT)
          .limit(Worker.ANZAHL_MESSWERTE)
          .boxed()
          .collect(Collectors.toList()))

      .parallel()
      .limit(ANZAHL_KURVEN)
      .collect(Collectors.toList())
      .stream()
      .parallel()
      .map(v -> new WorkLoadGenerator().generate(v))
      .collect(Collectors.toList());
}
```

Functions

how use Functions...

@SvenRuppert

Functions - Java8

Functions - Java8

@SvenRuppert

```
final List<String> names = Arrays.asList(
    "Hugo",
    "Willy",
    "Simon",
    "Erwin",
    "Sigfried"
);
```

Functions - Java8

@SvenRuppert

```
final List<String> names = Arrays.asList(
    "Hugo",
    "Willy",
    "Simon",
    "Erwin",
    "Sigfried"
);
```

```
names
    .stream()
    .filter(s -> s.contains("i"))
    .forEach(out::println);
```

Functions - Java8

@SvenRuppert

```
final List<String> names = Arrays.asList(
    "Hugo",
    "Willy",
    "Simon",
    "Erwin",
    "Sigfried"
);
```

```
names
    .stream()
    .filter(s -> s.contains("i"))
    .forEach(out::println);
```

```
names
    .stream()
    .filter(s -> s.contains("g"))
    .forEach(out::println);
```

Functions - Java8

@SvenRuppert

```
final List<String> names = Arrays.asList(
    "Hugo",
    "Willy",
    "Simon",
    "Erwin",
    "Sigfried"
);
```

```
names
    .stream()
    .filter(s -> s.contains("i"))
    .forEach(out::println);
```

```
names
    .stream()
    .filter(s -> s.contains("g"))
    .forEach(out::println);
```

Functions - Java8

@SvenRuppert

```
final List<String> names = Arrays.asList(
    "Hugo",
    "Willy",
    "Simon",
    "Erwin",
    "Sigfried"
);
```

```
names
    .stream()
    .filter(s -> s.contains("i"))
    .forEach(out::println);
```

```
names
    .stream()
    .filter(s -> s.contains("g"))
    .forEach(out::println);
```

Functions - Java8

@SvenRuppert

```
final List<String> names = Arrays.asList(
    "Hugo",
    "Willy",
    "Simon",
    "Erwin",
    "Sigfried"
);
```

Functions - Java8

@SvenRuppert

```
final List<String> names = Arrays.asList(
    "Hugo",
    "Willy",
    "Simon",
    "Erwin",
    "Sigfried"
);
```

```
final Consumer<String> println = out::println;
```

Functions - Java8

@SvenRuppert

```
final List<String> names = Arrays.asList(
    "Hugo",
    "Willy",
    "Simon",
    "Erwin",
    "Sigfried"
);
```

```
final Consumer<String> println = out::println;
```

```
final String v = "i";
```

Functions - Java8

@SvenRuppert

```
final List<String> names = Arrays.asList(
    "Hugo",
    "Willy",
    "Simon",
    "Erwin",
    "Sigfried"
);
```

```
final Consumer<String> println = out::println;
```

```
final String v = "i";
```

```
final Predicate<String> predicateI = s -> s.contains(v);
```

Functions - Java8

@SvenRuppert

```
final List<String> names = Arrays.asList(
    "Hugo",
    "Willy",
    "Simon",
    "Erwin",
    "Sigfried"
);
```

```
final Consumer<String> println = out::println;
```

```
final String v = "i";
```

```
final Predicate<String> predicateI = s -> s.contains(v);
```

```
names
    .stream()
    .filter(predicateI)
    .forEach(println);
```

Functions - Java8

@SvenRuppert

```
final List<String> names = Arrays.asList(
    "Hugo",
    "Willy",
    "Simon",
    "Erwin",
    "Sigfried"
);
```

```
final Consumer<String> println = out::println;
```

```
final String v = "i";
```

```
final Predicate<String> predicateI = s -> s.contains(v);
```

```
names
    .stream()
    .filter(predicateI)
    .forEach(println);
```

```
final Predicate<String> predicateG = s -> s.contains("g");
names
    .stream()
    .filter(predicateG)
    .forEach(println);
```

Functions - Java8

@SvenRuppert

```
final List<String> names = Arrays.asList(
    "Hugo",
    "Willy",
    "Simon",
    "Erwin",
    "Sigfried"
);
```

```
final Consumer<String> println = out::println;
```

```
final String v = "i";
```

```
final Predicate<String> predicateI = s -> s.contains(v);
```

```
names
    .stream()
    .filter(predicateI)
    .forEach(println);
```

```
final Predicate<String> predicateG = s -> s.contains("g");
names
    .stream()
    .filter(predicateG)
    .forEach(println);
```

Functions - Java8

@SvenRuppert

```
final List<String> names = Arrays.asList(
    "Hugo",
    "Willy",
    "Simon",
    "Erwin",
    "Sigfried"
);
```

```
final Consumer<String> println = out::println;
```

```
final String v = "i";
```

```
final Predicate<String> predicateI = s -> s.contains(v);
```

```
names
    .stream()
    .filter(predicateI)
    .forEach(println);
```

```
final Predicate<String> predicateG = s -> s.contains("g");
names
    .stream()
    .filter(predicateG)
    .forEach(println);
```

Functions - Java8

@SvenRuppert

```
final List<String> names = Arrays.asList(
    "Hugo",
    "Willy",
    "Simon",
    "Erwin",
    "Sigfried"
);
```

```
final Consumer<String> println = out::println;
```

```
final String v = "i";
```

```
final Predicate<String> predicateI = s -> s.contains(v);
```

```
names
    .stream()
    .filter(predicateI)
    .forEach(println);
```

```
final Predicate<String> predicateG = s -> s.contains("g");
names
    .stream()
    .filter(predicateG)
    .forEach(println);
```

Functions - Java8

@SvenRuppert

```
final Function<String, Predicate<String>> funPredicate
    = str -> (String s) -> s.contains(str);
```

Functions - Java8

@SvenRuppert

```
final Function<String, Predicate<String>> funPredicate
    = str -> (String s) -> s.contains(str);
```

Functions - Java8

@SvenRuppert

```
final Function<String, Predicate<String>> funPredicate
    = str -> (String s) -> s.contains(str);
```

Functions - Java8

@SvenRuppert

```
final Function<String, Predicate<String>> funPredicate
    = str -> (String s) -> s.contains(str);
```

```
names
    .stream()
    .filter(funPredicate.apply( t: "i"))
    .forEach(println);
```

Functions - Java8

@SvenRuppert

```
final Function<String, Predicate<String>> funPredicate
    = str -> (String s) -> s.contains(str);
```

```
names
    .stream()
    .filter(funPredicate.apply( t: "i"))
    .forEach(println);
```

```
names
    .stream()
    .filter(funPredicate.apply( t: "g"))
    .forEach(println);
```

Functions - Java8

@SvenRuppert

```
final Function<String, Predicate<String>> funPredicate
    = str -> (String s) -> s.contains(str);
```

```
names
    .stream()
    .filter(funPredicate.apply( t: "i"))
    .forEach(println);
```

```
names
    .stream()
    .filter(funPredicate.apply( t: "g"))
    .forEach(println);
```

Functions - Java8

Functions - Java8

@SvenRuppert

```
public void log(String msg) { System.out.println("msg = " + msg); }

public void logDebug(String msg) { System.out.println("msg = " + msg); }

public boolean isDebugEnabled() { return true; }
```

Functions - Java8

@SvenRuppert

```
public void log(String msg) { System.out.println("msg = " + msg); }

public void logDebug(String msg) { System.out.println("msg = " + msg); }

public boolean isDebugEnabled() { return true; }
```

```
public static final Supplier<String> logMessage = () -> {
  return "Hallo " + "Welt";
};
```

Functions - Java8

@SvenRuppert

```
public void log(String msg) { System.out.println("msg = " + msg); }

public void logDebug(String msg) { System.out.println("msg = " + msg); }

public boolean isDebugEnabled() { return true; }
```

```
public static final Supplier<String> logMessage = () -> {
  return "Hallo " + "Welt";
};
```

```
public void log_001(Supplier<String> msg) {
  if (isDebugEnabled())
    System.out.println("msg = " + msg.get());
}
```

Functions - Java8

@SvenRuppert

```
public static final Supplier<String> logMessage = () -> {
  return "Hallo " + "Welt";
};
```

Functions - Java8

@SvenRuppert

```
public static final Supplier<String> logMessage = () -> {
  return "Hallo " + "Welt";
};
```

```
main.log( msg: "Hallo " + "Welt");
main.logDebug( msg: "Hallo " + "Welt");
if (main.isDebugEnabled()) {
  main.logDebug( msg: "Hallo " + "Welt");
}
main.log_001(logMessage);
main.log_001(() -> "All bad " + "pppffff");
```

Functions - Java8

@SvenRuppert

```
public static final Supplier<String> logMessage = () -> {
  return "Hallo " + "Welt";
};
```

Functions - Java8

@SvenRuppert

```java
public static final Supplier<String> logMessage = () -> {
  return "Hallo " + "Welt";
};
```

```java
public static final Supplier<String> logMessageHello = () -> {
  return "Hallo ";
};
public static final Supplier<String> logMessageWorld = () -> {
  return "Welt";
};
```

Functions - Java8

@SvenRuppert

```
public static final Supplier<String> logMessageHello = () -> {
  return "Hallo ";
};
public static final Supplier<String> logMessageWorld = () -> {
  return "Welt";
};
```

Functions - Java8

@SvenRuppert

```
public static final Supplier<String> logMessageHello = () -> {
  return "Hallo ";
};
public static final Supplier<String> logMessageWorld = () -> {
  return "Welt";
};
```

```
public static final Supplier<String> logMessageHelloWorld = () -> {
  return logMessageHello.get() + logMessageWorld.get();
};
```

Functions - Java8

@SvenRuppert

```
public static final Supplier<Supplier<String>> logMessageHelloWorldSupplier
    = () -> () -> logMessageHello.get() + logMessageWorld.get();
```

Functions - Java8

@SvenRuppert

```
public static final Supplier<String> logMessageHello = () -> {
  return "Hallo ";
};
public static final Supplier<String> logMessageWorld = () -> {
  return "Welt";
};
```

```
public static final Supplier<Supplier<String>> logMessageHelloWorldSupplier
    = () -> () -> logMessageHello.get() + logMessageWorld.get();
```

Functions - Java8

@SvenRuppert

```
public static final Supplier<String> logMessageHello = () -> {
  return "Hallo ";
};
public static final Supplier<String> logMessageWorld = () -> {
  return "Welt";
};
```

```
public static final Supplier<Supplier<String>> logMessageHelloWorldSupplier
    = () -> () -> logMessageHello.get() + logMessageWorld.get();
```

```
public void log_002(Supplier<Supplier<String>> msg) {
  if (isDebugEnabled())
    System.out.println("msg = " + msg.get());
}
```

Functions - Java8

@SvenRuppert

```
public static final Supplier<String> logMessageHello = () -> {
  return "Hallo ";
};
public static final Supplier<String> logMessageWorld = () -> {
  return "Welt";
};
```

```
public static final Supplier<Supplier<String>> logMessageHelloWorldSupplier
    = () -> () -> logMessageHello.get() + logMessageWorld.get();
```

```
public void log_002(Supplier<Supplier<String>> msg) {
  if (isDebugEnabled())
    System.out.println("msg = " + msg.get());
}
```

Reactive stuff

how use Functions... async...

@SvenRuppert

Observer

Observer

@SvenRuppert

```
public static class Observable<KEY, VALUE> {

  private final Map<KEY, Consumer<VALUE>> listeners = new ConcurrentHashMap<>();

  public void register(KEY key, Consumer<VALUE> listener) { listeners.put(key, listener); }

  public void unregister(KEY key) { listeners.remove(key); }

  public void sendEvent(VALUE event) { listeners.values().forEach(listener -> listener.accept(event)); }

}
```

Observer

@SvenRuppert

```
public static class Observable<KEY, VALUE> {

  private final Map<KEY, Consumer<VALUE>> listeners = new ConcurrentHashMap<>();

  public void register(KEY key, Consumer<VALUE> listener) { listeners.put(key, listener); }

  public void unregister(KEY key) { listeners.remove(key); }

  public void sendEvent(VALUE event) { listeners.values().forEach(listener -> listener.accept(event)); }

}
```

Observer

@SvenRuppert

```
public static class Observable<KEY, VALUE> {

  private final Map<KEY, Consumer<VALUE>> listeners = new ConcurrentHashMap<>();

  public void register(KEY key, Consumer<VALUE> listener) { listeners.put(key, listener); }

  public void unregister(KEY key) { listeners.remove(key); }

  public void sendEvent(VALUE event) { listeners.values().forEach(listener -> listener.accept(event)); }

}
```

Observer

@SvenRuppert

```
public static class Observable<KEY, VALUE> {

  private final Map<KEY, Consumer<VALUE>> listeners = new ConcurrentHashMap<>();

  public void register(KEY key, Consumer<VALUE> listener) { listeners.put(key, listener); }

  public void unregister(KEY key) { listeners.remove(key); }

  public void sendEvent(VALUE event) { listeners.values().forEach(listener -> listener.accept(event)); }

}
```

```
public static void main(String[] args) {
  final Observable<String, String> observable = new Observable<>();
  observable.register( key: "key1", System.out::println);
  observable.register( key: "key2", System.out::println);

  observable.sendEvent( event: "Hello World!");

  observable.unregister( key: "key1");
  observable.sendEvent( event: "Hello World!");
}
```

Java8 - Concurrency - CompletableFuture

Process

Java8 - Concurrency - CompletableFuture

Process

Thread

Java8 - Concurrency - CompletableFuture

Process

Thread

Fiber

Java8 - Concurrency - CompletableFuture

@SvenRuppert

Process

Thread

Fiber

Java8 - Concurrency - CompletableFuture

Process

Thread

Fiber

CompletionStage

Java8 - Concurrency - CompletableFuture

Process

Thread

Fiber

CompletionStage

Java8 - Concurrency - CompletableFuture

@SvenRuppert

Process

Thread

Fiber

CompletionStage

Java8 - Concurrency - CompletableFuture

@SvenRuppert

Process

Thread

Fiber

CompletionStage CompletionStage

Java8 - Concurrency - CompletableFuture

@SvenRuppert

Process

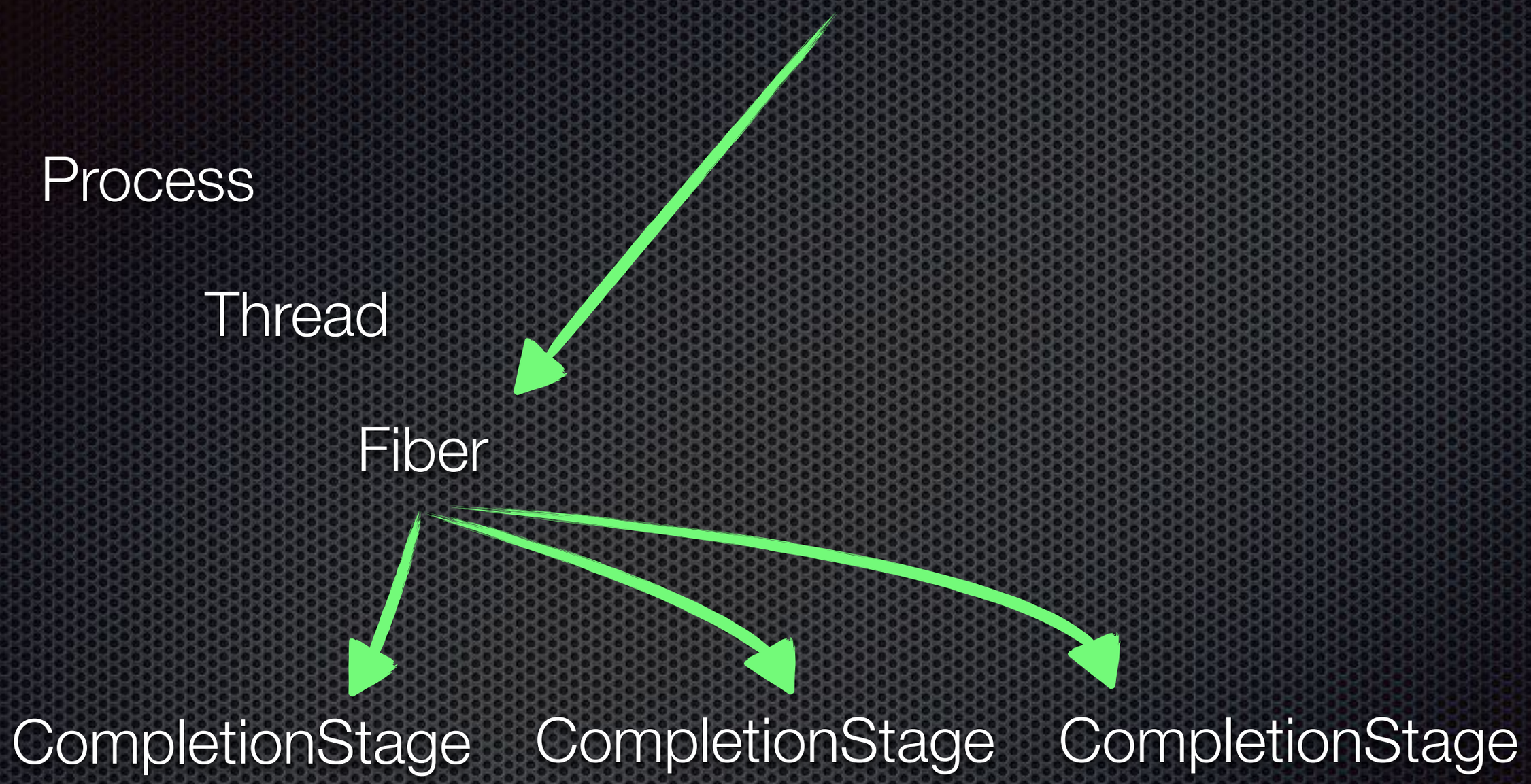
Thread

Fiber

CompletionStage CompletionStage

Java8 - Concurrency - CompletableFuture

@SvenRuppert



Java8 - Concurrency - CompletableFuture

@SvenRuppert

```
final ExecutorService fixedThreadPool = Executors
    .newFixedThreadPool(
        Runtime.getRuntime().availableProcessors());

final Supplier<String> task = () -> "das Ergebnis";

final CompletableFuture<String> future
    = CompletableFuture.supplyAsync(task, fixedThreadPool);

final CompletableFuture<Void> accept = future
    .thenAccept(System.out::println);
```

Java8 - Concurrency - CompletableFuture

Value A

Java8 - Concurrency - CompletableFuture

@SvenRuppert

Value A Value B

Java8 - Concurrency - CompletableFuture

Value A Value B

Java8 - Concurrency - CompletableFuture

@SvenRuppert

Value A Value B

Java8 - Concurrency - CompletableFuture

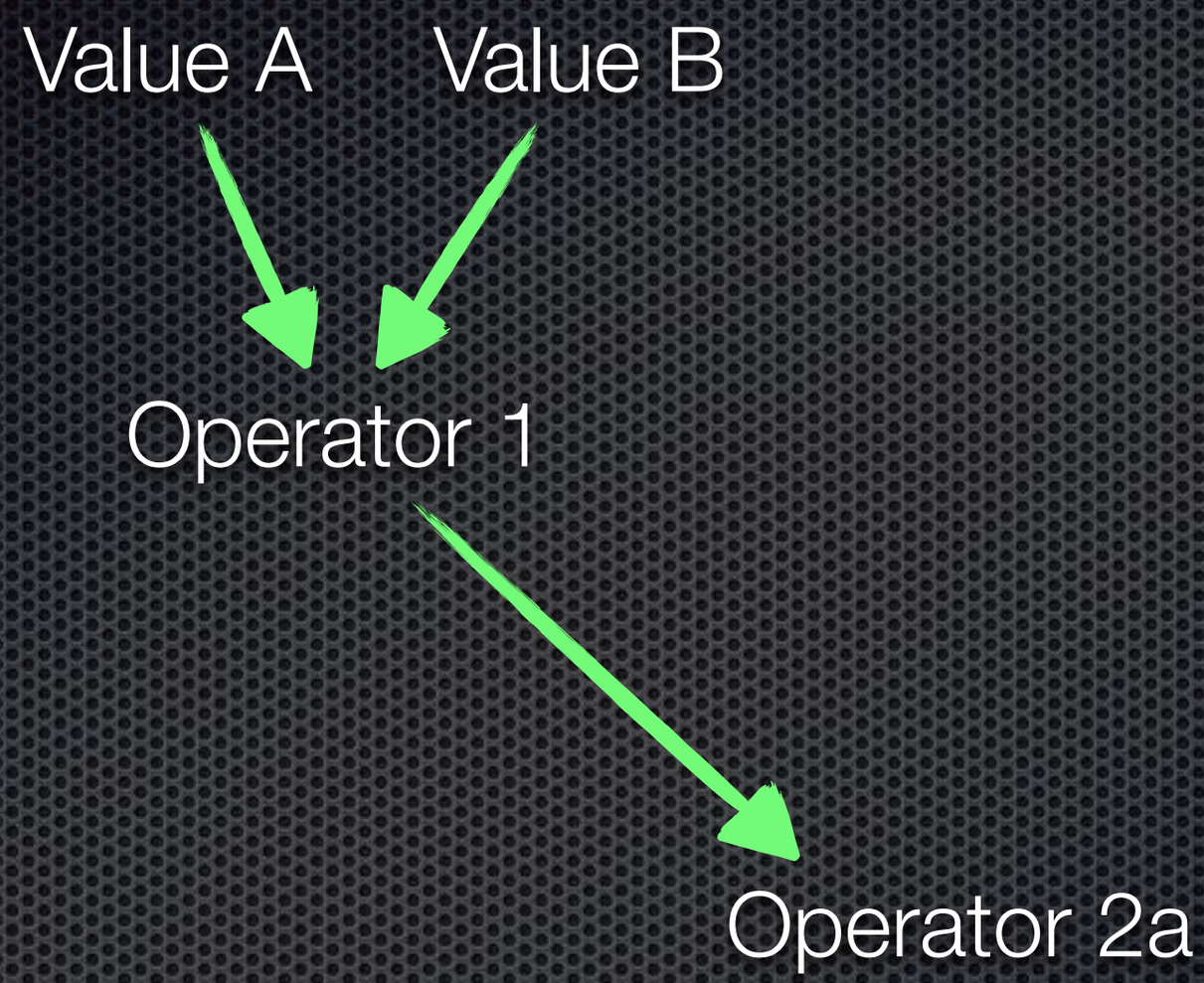
@SvenRuppert

Java8 - Concurrency - CompletableFuture

@SvenRuppert

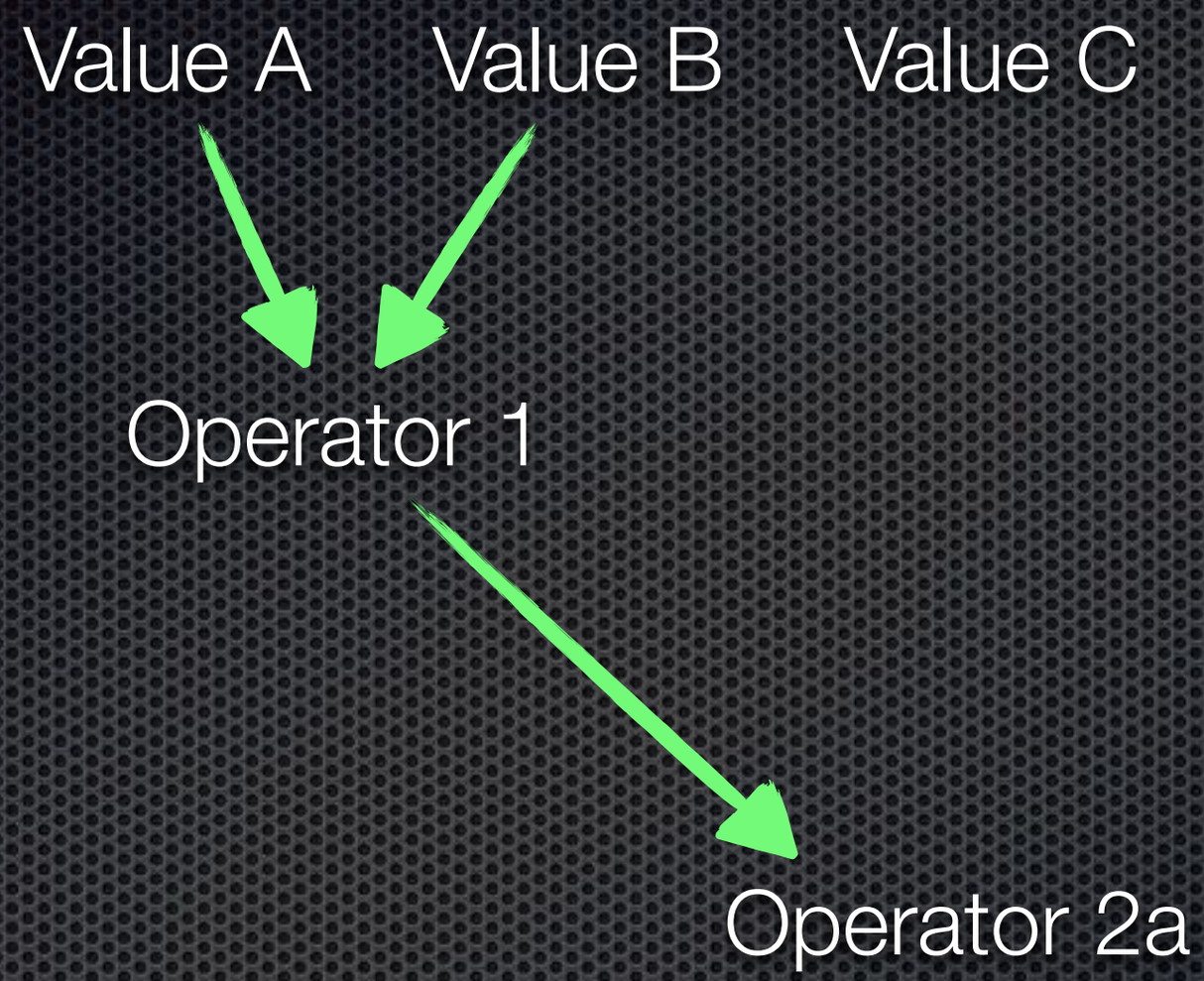
Java8 - Concurrency - CompletableFuture

@SvenRuppert



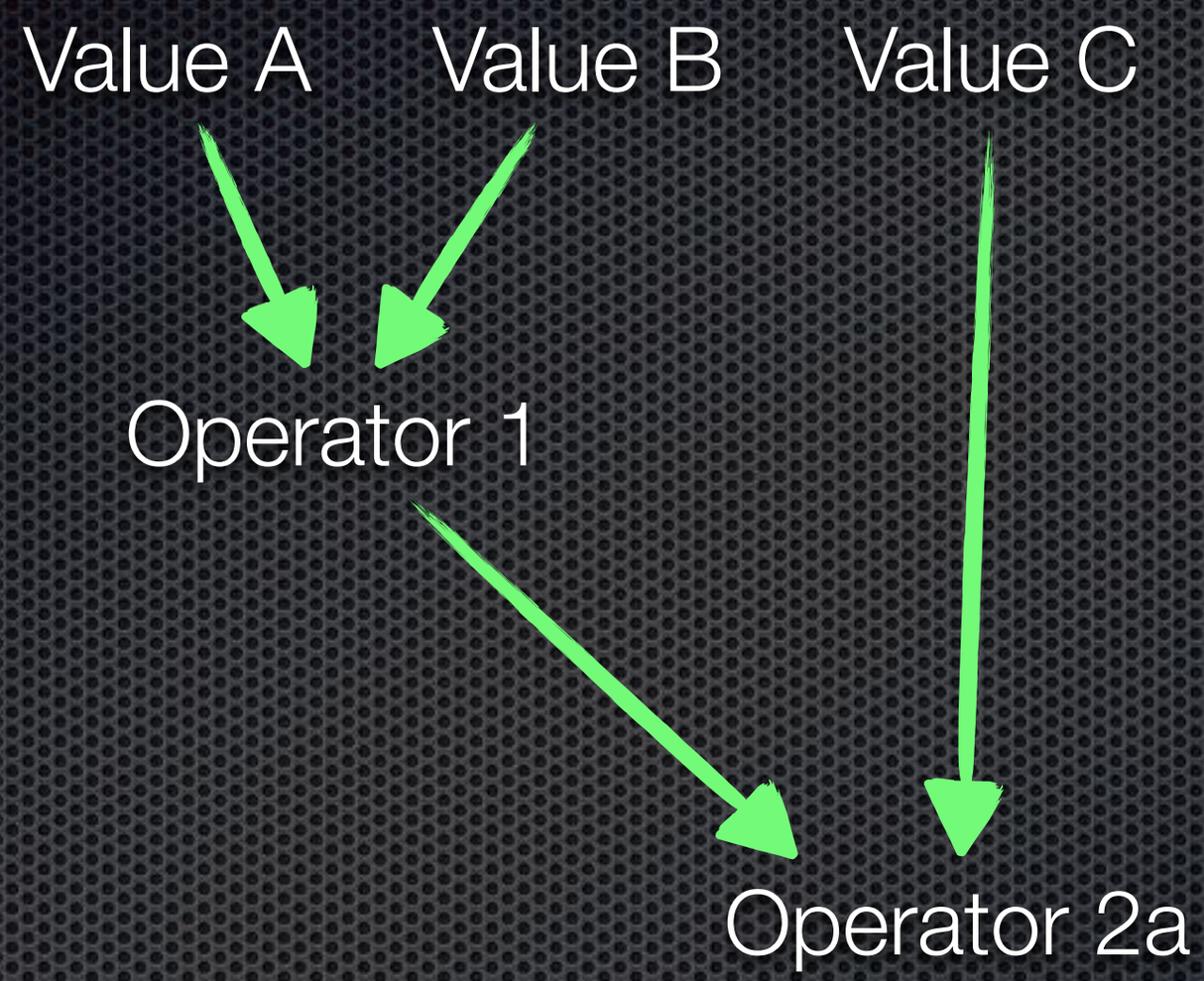
Java8 - Concurrency - CompletableFuture

@SvenRuppert



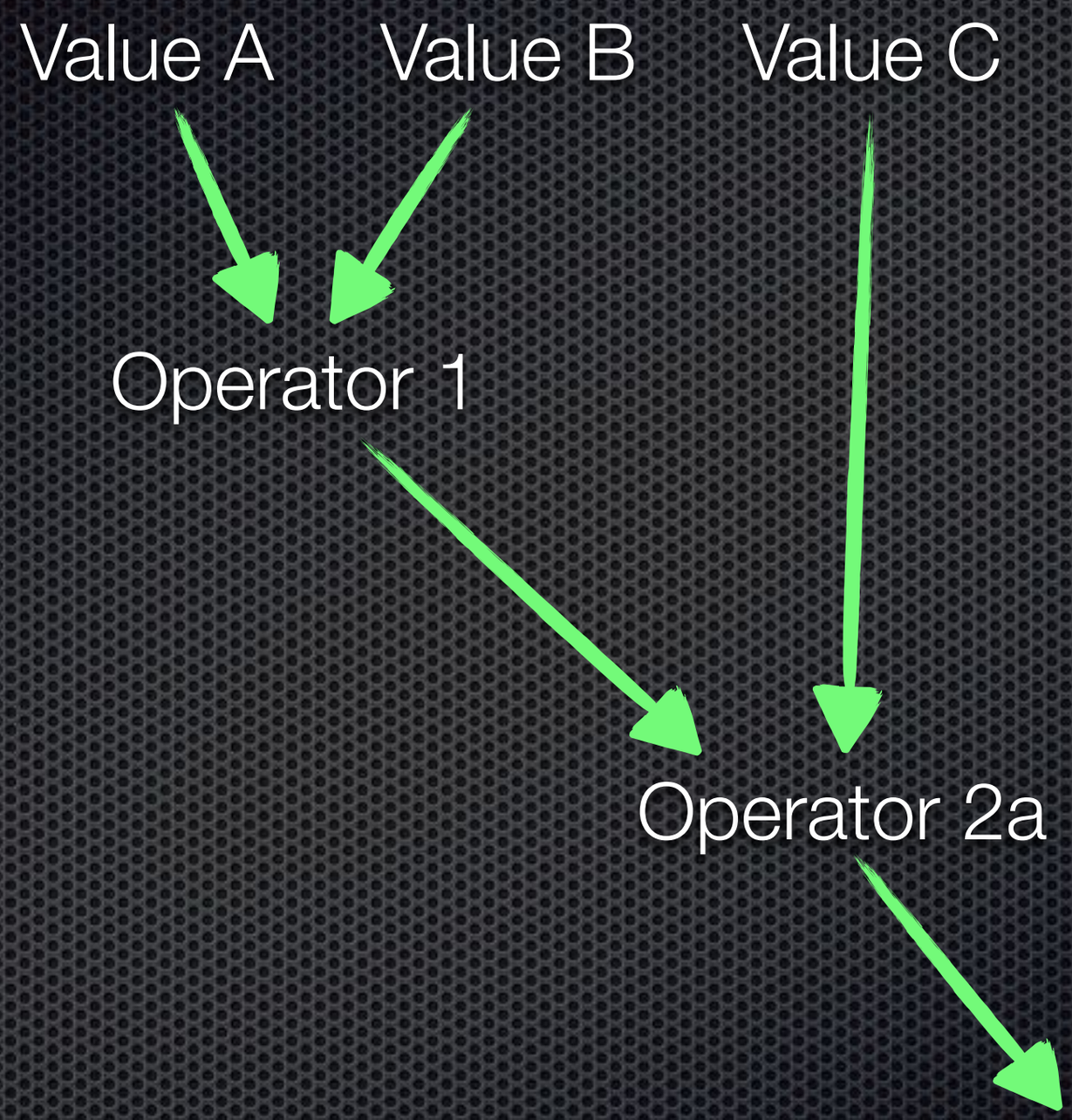
Java8 - Concurrency - CompletableFuture

@SvenRuppert



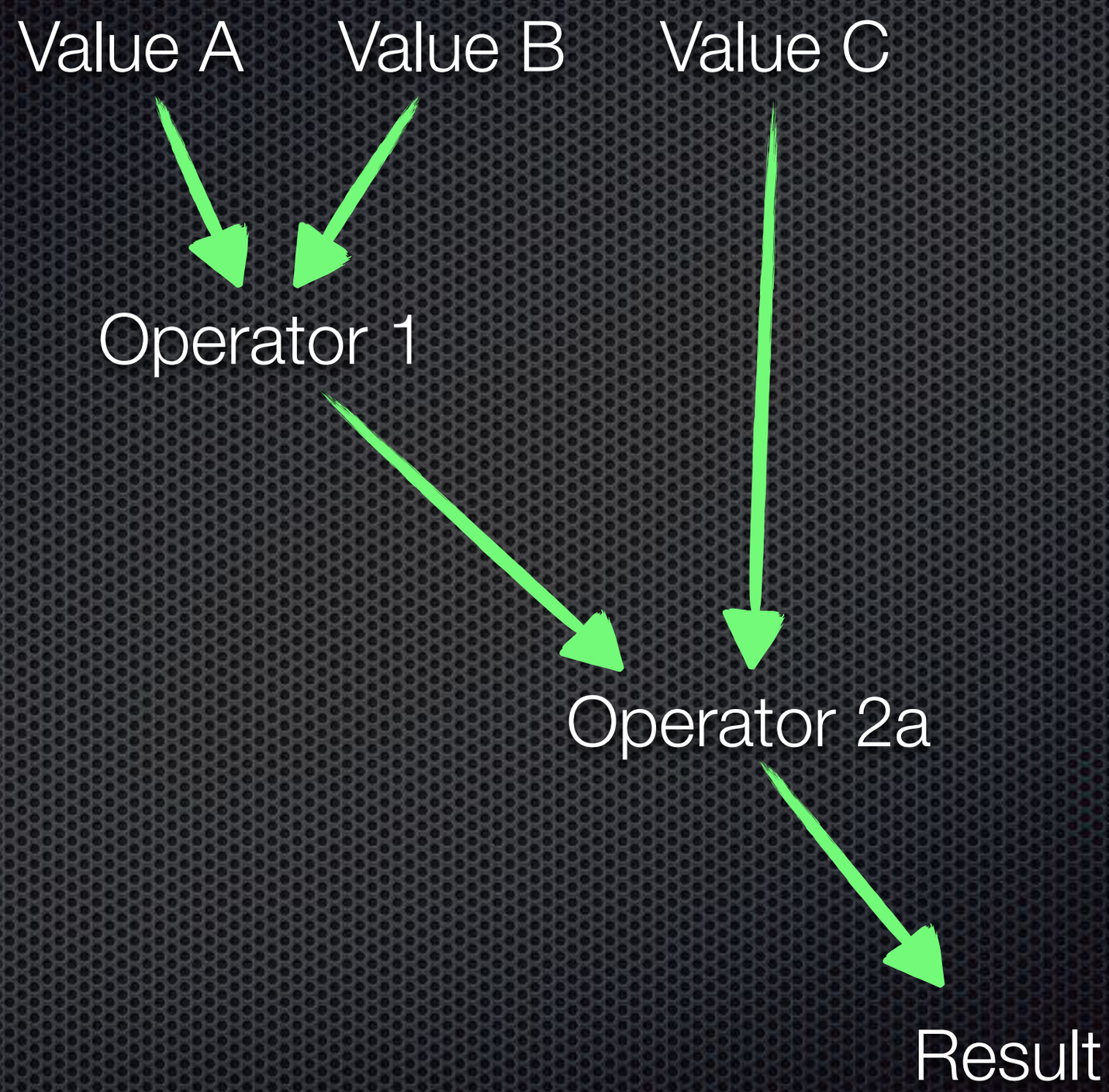
Java8 - Concurrency - CompletableFuture

@SvenRuppert



Java8 - Concurrency - CompletableFuture

@SvenRuppert



Java8 - Concurrency - CompletableFuture

@SvenRuppert

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```
final int availableProcessors = Runtime.getRuntime().availableProcessors();
final ExecutorService executorServiceCPU = Executors.newWorkStealingPool(availableProcessors);
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```
final int availableProcessors = Runtime.getRuntime().availableProcessors();
final ExecutorService executorServiceCPU = Executors.newWorkStealingPool(availableProcessors);
```

```
final CompletableFuture<StepA> stepA = CompletableFuture
    .supplyAsync(new Supplier<StepA>() {
      @Override
      public StepA get() {
        // do some long running stuff
        final StepA step = new StepA();
        step.message = "Step A done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```
final int availableProcessors = Runtime.getRuntime().availableProcessors();
final ExecutorService executorServiceCPU = Executors.newWorkStealingPool(availableProcessors);
```

```
final CompletableFuture<StepA> stepA = CompletableFuture
    .supplyAsync(new Supplier<StepA>() {
      @Override
      public StepA get() {
        // do some long running stuff
        final StepA step = new StepA();
        step.message = "Step A done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```java
final int availableProcessors = Runtime.getRuntime().availableProcessors();
final ExecutorService executorServiceCPU = Executors.newWorkStealingPool(availableProcessors);
```

```java
final CompletableFuture<StepA> stepA = CompletableFuture
    .supplyAsync(new Supplier<StepA>() {
      @Override
      public StepA get() {
        // do some long running stuff
        final StepA step = new StepA();
        step.message = "Step A done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<StepB> stepB = CompletableFuture
    .supplyAsync(new Supplier<StepB>() {
      @Override
      public StepB get() {
        // do some stuff
        final StepB step = new StepB();
        step.message = "Step B done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```java
final int availableProcessors = Runtime.getRuntime().availableProcessors();
final ExecutorService executorServiceCPU = Executors.newWorkStealingPool(availableProcessors);
```

```java
final CompletableFuture<StepA> stepA = CompletableFuture
    .supplyAsync(new Supplier<StepA>() {
      @Override
      public StepA get() {
        // do some long running stuff
        final StepA step = new StepA();
        step.message = "Step A done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<StepB> stepB = CompletableFuture
    .supplyAsync(new Supplier<StepB>() {
      @Override
      public StepB get() {
        // do some stuff
        final StepB step = new StepB();
        step.message = "Step B done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<StepC> stepC = CompletableFuture
    .supplyAsync(new Supplier<StepC>() {
      @Override
      public StepC get() {
        // do some stuff
        final StepC step = new StepC();
        step.message = "Step C done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```java
final int availableProcessors = Runtime.getRuntime().availableProcessors();
final ExecutorService executorServiceCPU = Executors.newWorkStealingPool(availableProcessors);
```

```java
final CompletableFuture<StepA> stepA = CompletableFuture
    .supplyAsync(new Supplier<StepA>() {
      @Override
      public StepA get() {
        // do some long running stuff
        final StepA step = new StepA();
        step.message = "Step A done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<StepB> stepB = CompletableFuture
    .supplyAsync(new Supplier<StepB>() {
      @Override
      public StepB get() {
        // do some stuff
        final StepB step = new StepB();
        step.message = "Step B done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<StepC> stepC = CompletableFuture
    .supplyAsync(new Supplier<StepC>() {
      @Override
      public StepC get() {
        // do some stuff
        final StepC step = new StepC();
        step.message = "Step C done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```java
final int availableProcessors = Runtime.getRuntime().availableProcessors();
final ExecutorService executorServiceCPU = Executors.newWorkStealingPool(availableProcessors);
```

```java
final CompletableFuture<StepA> stepA = CompletableFuture
    .supplyAsync(new Supplier<StepA>() {
      @Override
      public StepA get() {
        // do some long running stuff
        final StepA step = new StepA();
        step.message = "Step A done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<StepB> stepB = CompletableFuture
    .supplyAsync(new Supplier<StepB>() {
      @Override
      public StepB get() {
        // do some stuff
        final StepB step = new StepB();
        step.message = "Step B done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<StepC> stepC = CompletableFuture
    .supplyAsync(new Supplier<StepC>() {
      @Override
      public StepC get() {
        // do some stuff
        final StepC step = new StepC();
        step.message = "Step C done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<Result> stepAstepBCombined = stepA
    .thenCombineAsync(stepB, new BiFunction<StepA, StepB, Result>() {
      @Override
      public Result apply(final StepA stepA, final StepB stepB) {
        final Result result = new Result();
        result.message = stepA.message + " - " + stepB.message;
        System.out.println("result = " + result);
        return result;
      }
    });
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```java
final int availableProcessors = Runtime.getRuntime().availableProcessors();
final ExecutorService executorServiceCPU = Executors.newWorkStealingPool(availableProcessors);
```

```java
final CompletableFuture<StepA> stepA = CompletableFuture
    .supplyAsync(new Supplier<StepA>() {
      @Override
      public StepA get() {
        // do some long running stuff
        final StepA step = new StepA();
        step.message = "Step A done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<StepB> stepB = CompletableFuture
    .supplyAsync(new Supplier<StepB>() {
      @Override
      public StepB get() {
        // do some stuff
        final StepB step = new StepB();
        step.message = "Step B done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<StepC> stepC = CompletableFuture
    .supplyAsync(new Supplier<StepC>() {
      @Override
      public StepC get() {
        // do some stuff
        final StepC step = new StepC();
        step.message = "Step C done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<Result> stepAstepBCombined = stepA
    .thenCombineAsync(stepB, new BiFunction<StepA, StepB, Result>() {
      @Override
      public Result apply(final StepA stepA, final StepB stepB) {
        final Result result = new Result();
        result.message = stepA.message + " - " + stepB.message;
        System.out.println("result = " + result);
        return result;
      }
    });
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```java
final int availableProcessors = Runtime.getRuntime().availableProcessors();
final ExecutorService executorServiceCPU = Executors.newWorkStealingPool(availableProcessors);
```

```java
final CompletableFuture<StepA> stepA = CompletableFuture
    .supplyAsync(new Supplier<StepA>() {
      @Override
      public StepA get() {
        // do some long running stuff
        final StepA step = new StepA();
        step.message = "Step A done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<StepB> stepB = CompletableFuture
    .supplyAsync(new Supplier<StepB>() {
      @Override
      public StepB get() {
        // do some stuff
        final StepB step = new StepB();
        step.message = "Step B done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<StepC> stepC = CompletableFuture
    .supplyAsync(new Supplier<StepC>() {
      @Override
      public StepC get() {
        // do some stuff
        final StepC step = new StepC();
        step.message = "Step C done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<Result> stepAstepBCombined = stepA
    .thenCombineAsync(stepB, new BiFunction<StepA, StepB, Result>() {
      @Override
      public Result apply(final StepA stepA, final StepB stepB) {
        final Result result = new Result();
        result.message = stepA.message + " - " + stepB.message;
        System.out.println("result = " + result);
        return result;
      }
    });
```

```java
final CompletableFuture<Result> stepABstepCCombined = stepC
    .thenCombineAsync(stepAstepBCombined, new BiFunction<StepC, Result, Result>() {
      @Override
      public Result apply(final StepC stepC, final Result result) {
        final Result resultC = new Result();
        resultC.message = stepC.message + " - " + result.message;
        System.out.println("result = " + resultC);
        return resultC;
      }
    });
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```java
final int availableProcessors = Runtime.getRuntime().availableProcessors();
final ExecutorService executorServiceCPU = Executors.newWorkStealingPool(availableProcessors);
```

```java
final CompletableFuture<StepA> stepA = CompletableFuture
    .supplyAsync(new Supplier<StepA>() {
      @Override
      public StepA get() {
        // do some long running stuff
        final StepA step = new StepA();
        step.message = "Step A done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<StepB> stepB = CompletableFuture
    .supplyAsync(new Supplier<StepB>() {
      @Override
      public StepB get() {
        // do some stuff
        final StepB step = new StepB();
        step.message = "Step B done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<StepC> stepC = CompletableFuture
    .supplyAsync(new Supplier<StepC>() {
      @Override
      public StepC get() {
        // do some stuff
        final StepC step = new StepC();
        step.message = "Step C done";
        System.out.println("step = " + step);
        return step;
      }
    }, executorServiceCPU);
```

```java
final CompletableFuture<Result> stepAstepBCombined = stepA
    .thenCombineAsync(stepB, new BiFunction<StepA, StepB, Result>() {
      @Override
      public Result apply(final StepA stepA, final StepB stepB) {
        final Result result = new Result();
        result.message = stepA.message + " - " + stepB.message;
        System.out.println("result = " + result);
        return result;
      }
    });
```

```java
final CompletableFuture<Result> stepABstepCCombined = stepC
    .thenCombineAsync(stepAstepBCombined, new BiFunction<StepC, Result, Result>() {
      @Override
      public Result apply(final StepC stepC, final Result result) {
        final Result resultC = new Result();
        resultC.message = stepC.message + " - " + result.message;
        System.out.println("result = " + resultC);
        return resultC;
      }
    });
```

```java
stepABstepCCombined.thenAccept(System.out::println);
```

Java8 - Concurrency - CompletableFuture

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```
final int availableProcessors = Runtime.getRuntime().availableProcessors();

final ExecutorService executorServiceCPU = Executors
    .newWorkStealingPool(availableProcessors);

final ExecutorService executorServiceNetworkLocalDisk = Executors
    .newWorkStealingPool(availableProcessors * 10);

final ExecutorService executorServiceNetworkRest = Executors
    .newWorkStealingPool(availableProcessors * 20);
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```java
final int availableProcessors = Runtime.getRuntime().availableProcessors();

final ExecutorService executorServiceCPU = Executors
    .newWorkStealingPool(availableProcessors);

final ExecutorService executorServiceNetworkLocalDisk = Executors
    .newWorkStealingPool(availableProcessors * 10);

final ExecutorService executorServiceNetworkRest = Executors
    .newWorkStealingPool(availableProcessors * 20);
```

```java
final BiFunction<StepA, StepB, Result> stepAStepBResultBiFunction = (stepA, stepB) -> {
  final Result result = new Result();
  result.message = stepA.message + " - " + stepB.message + " merged AB";
  System.out.println("result = " + result);
  return result;
};

final BiFunction<StepC, Result, Result> stepCResultResultBiFunction = (stepC, result) -> {
  final Result resultC = new Result();
  resultC.message = stepC.message + " - " + result.message + " merged AB c" ;
  System.out.println("result = " + resultC);
  return resultC;
};
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```java
final int availableProcessors = Runtime.getRuntime().availableProcessors();

final ExecutorService executorServiceCPU = Executors
    .newWorkStealingPool(availableProcessors);

final ExecutorService executorServiceNetworkLocalDisk = Executors
    .newWorkStealingPool(availableProcessors * 10);

final ExecutorService executorServiceNetworkRest = Executors
    .newWorkStealingPool(availableProcessors * 20);
```

```java
final BiFunction<StepA, StepB, Result> stepAStepBResultBiFunction = (stepA, stepB) -> {
  final Result result = new Result();
  result.message = stepA.message + " - " + stepB.message + " merged AB";
  System.out.println("result = " + result);
  return result;
};

final BiFunction<StepC, Result, Result> stepCResultResultBiFunction = (stepC, result) -> {
  final Result resultC = new Result();
  resultC.message = stepC.message + " - " + result.message + " merged AB c" ;
  System.out.println("result = " + resultC);
  return resultC;
};
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```java
final int availableProcessors = Runtime.getRuntime().availableProcessors();

final ExecutorService executorServiceCPU = Executors
    .newWorkStealingPool(availableProcessors);

final ExecutorService executorServiceNetworkLocalDisk = Executors
    .newWorkStealingPool(availableProcessors * 10);

final ExecutorService executorServiceNetworkRest = Executors
    .newWorkStealingPool(availableProcessors * 20);
```

```java
final BiFunction<StepA, StepB, Result> stepAStepBResultBiFunction = (stepA, stepB) -> {
  final Result result = new Result();
  result.message = stepA.message + " - " + stepB.message + " merged AB";
  System.out.println("result = " + result);
  return result;
};

final BiFunction<StepC, Result, Result> stepCResultResultBiFunction = (stepC, result) -> {
  final Result resultC = new Result();
  resultC.message = stepC.message + " - " + result.message + " merged AB c" ;
  System.out.println("result = " + resultC);
  return resultC;
};
```

```java
final CompletableFuture<StepA> stepACF = CompletableFuture.supplyAsync(stepASupplier, executorServiceCPU);
final CompletableFuture<StepB> stepBCF = CompletableFuture.supplyAsync(stepBSupplier, executorServiceNetworkLocalDisk);
final CompletableFuture<Result> stepAstepBCombined = stepACF.thenCombineAsync(stepBCF, stepAStepBResultBiFunction); // CPU Pool


final CompletableFuture<StepC> stepCCF = CompletableFuture.supplyAsync(stepCSupplier, executorServiceNetworkRest);
final CompletableFuture<Result> stepABstepCCombined = stepCCF.thenCombineAsync(stepAstepBCombined, stepCResultResultBiFunction);
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```java
final int availableProcessors = Runtime.getRuntime().availableProcessors();

final ExecutorService executorServiceCPU = Executors
    .newWorkStealingPool(availableProcessors);

final ExecutorService executorServiceNetworkLocalDisk = Executors
    .newWorkStealingPool(availableProcessors * 10);

final ExecutorService executorServiceNetworkRest = Executors
    .newWorkStealingPool(availableProcessors * 20);
```

```java
final BiFunction<StepA, StepB, Result> stepAStepBResultBiFunction = (stepA, stepB) -> {
  final Result result = new Result();
  result.message = stepA.message + " - " + stepB.message + " merged AB";
  System.out.println("result = " + result);
  return result;
};

final BiFunction<StepC, Result, Result> stepCResultResultBiFunction = (stepC, result) -> {
  final Result resultC = new Result();
  resultC.message = stepC.message + " - " + result.message + " merged AB c" ;
  System.out.println("result = " + resultC);
  return resultC;
};
```

```java
final CompletableFuture<StepA> stepACF = CompletableFuture.supplyAsync(stepASupplier, executorServiceCPU);
final CompletableFuture<StepB> stepBCF = CompletableFuture.supplyAsync(stepBSupplier, executorServiceNetworkLocalDisk);
final CompletableFuture<Result> stepAstepBCombined = stepACF.thenCombineAsync(stepBCF, stepAStepBResultBiFunction); // CPU Pool


final CompletableFuture<StepC> stepCCF = CompletableFuture.supplyAsync(stepCSupplier, executorServiceNetworkRest);
final CompletableFuture<Result> stepABstepCCombined = stepCCF.thenCombineAsync(stepAstepBCombined, stepCResultResultBiFunction);
```

```java
stepABstepCCombined.thenAccept(System.out::println);
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```
final CompletableFuture<StepA> stepACF = CompletableFuture.supplyAsync(stepASupplier, executorServiceCPU);
final CompletableFuture<StepB> stepBCF = CompletableFuture.supplyAsync(stepBSupplier, executorServiceNetworkLocalDisk);
final CompletableFuture<Result> stepAstepBCombined = stepACF.thenCombineAsync(stepBCF, stepAStepBResultBiFunction); // CPU Pool


final CompletableFuture<StepC> stepCCF = CompletableFuture.supplyAsync(stepCSupplier, executorServiceNetworkRest);
final CompletableFuture<Result> stepABstepCCombined = stepCCF.thenCombineAsync(stepAstepBCombined, stepCResultResultBiFunction);
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```
final CompletableFuture<StepA> stepACF = CompletableFuture.supplyAsync(stepASupplier, executorServiceCPU);
final CompletableFuture<StepB> stepBCF = CompletableFuture.supplyAsync(stepBSupplier, executorServiceNetworkLocalDisk);
final CompletableFuture<Result> stepAstepBCombined = stepACF.thenCombineAsync(stepBCF, stepAStepBResultBiFunction); // CPU Pool


final CompletableFuture<StepC> stepCCF = CompletableFuture.supplyAsync(stepCSupplier, executorServiceNetworkRest);
final CompletableFuture<Result> stepABstepCCombined = stepCCF.thenCombineAsync(stepAstepBCombined, stepCResultResultBiFunction);
```

```
final CompletableFuture<StepA> stepACF = supplyAsync(MainLazy::getStepASupplier, executorServiceCPU);
final CompletableFuture<StepB> stepBCF = supplyAsync(MainLazy::getStepBSupplier, executorServiceNetworkLocalDisk);
```

Java8 - Concurrency - CompletableFuture

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```
// Operator
private static BiFunction<StepC, Result, Result> stepCResultResultBiFunction = (stepC, result) -> {
  final Result resultC = new Result();
  resultC.message = stepC.message + " - " + result.message + " merged AB c";
  System.out.println("result = " + resultC);
  return resultC;
};
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

```java
// Operator
private static BiFunction<StepC, Result, Result> stepCResultResultBiFunction = (stepC, result) -> {
  final Result resultC = new Result();
  resultC.message = stepC.message + " - " + result.message + " merged AB c";
  System.out.println("result = " + resultC);
  return resultC;
};
```

```java
public static BiFunction<StepA, StepB, Result> stepAStepBResultBiFunction = (stepA, stepB) -> {
  final Result result = new Result();
  result.message = stepA.message + " - " + stepB.message + " merged AB";
  System.out.println("result = " + result);
  return result;
};
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert

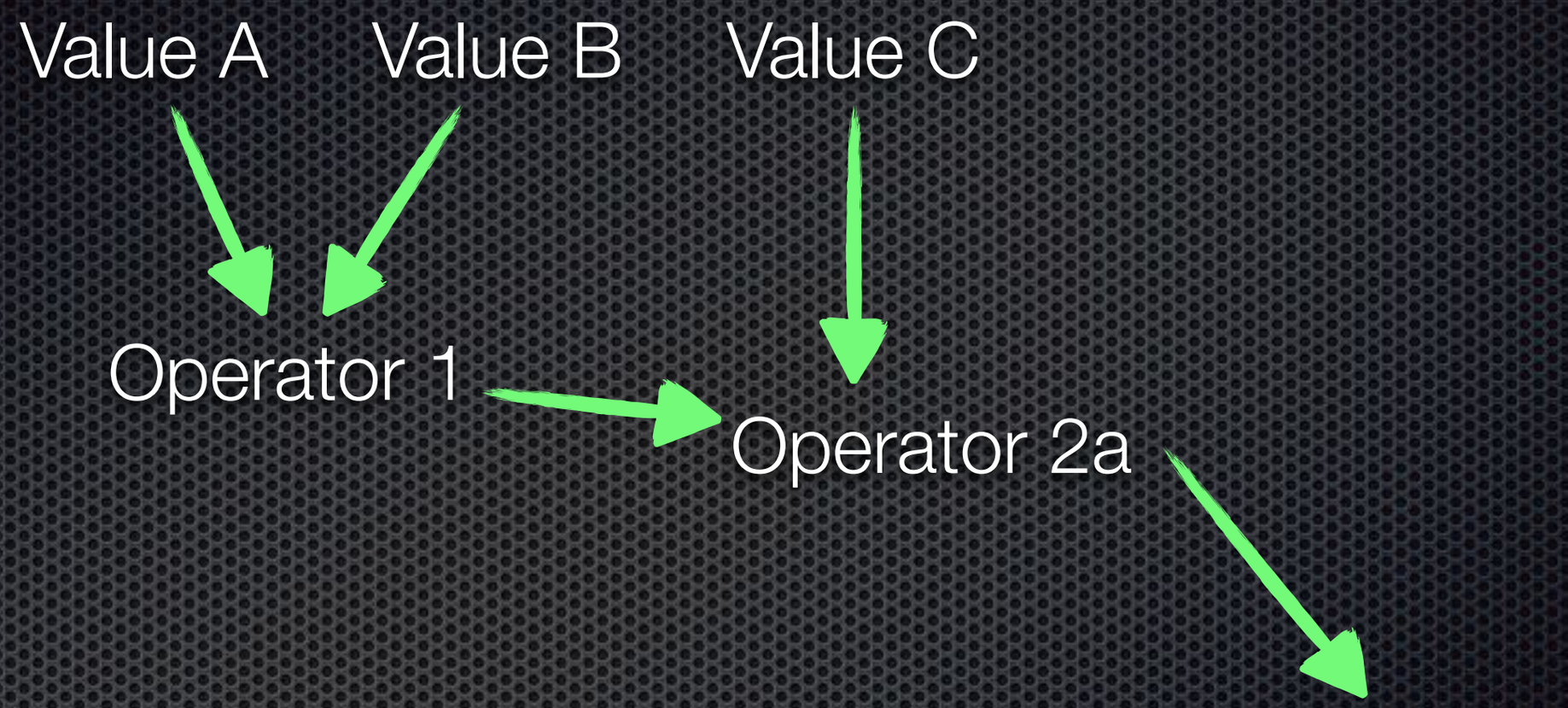
```java
// Operator
private static BiFunction<StepC, Result, Result> stepCResultResultBiFunction = (stepC, result) -> {
  final Result resultC = new Result();
  resultC.message = stepC.message + " - " + result.message + " merged AB c";
  System.out.println("result = " + resultC);
  return resultC;
};
```

```java
public static BiFunction<StepA, StepB, Result> stepAStepBResultBiFunction = (stepA, stepB) -> {
  final Result result = new Result();
  result.message = stepA.message + " - " + stepB.message + " merged AB";
  System.out.println("result = " + result);
  return result;
};
```

```java
supplyAsync(MainLazy::getStepCSupplier, executorServiceNetworkRest)
    .thenCombineAsync(supplyAsync(MainLazy::getStepASupplier, executorServiceCPU)
            .thenCombineAsync(supplyAsync(MainLazy::getStepBSupplier, executorServiceLocalDisk),
                stepAStepBResultBiFunction),
        stepCResultResultBiFunction)
    .thenAccept(System.out::println);
```

Java8 - Concurrency - CompletableFuture

@SvenRuppert




```
supplyAsync(MainLazy::getStepCSupplier, executorServiceNetworkRest)
    .thenCombineAsync(supplyAsync(MainLazy::getStepASupplier, executorServiceCPU)
            .thenCombineAsync(supplyAsync(MainLazy::getStepBSupplier, executorServiceLocalDisk),
                stepAStepBResultBiFunction),
        stepCResultResultBiFunction)
    .thenAccept(System.out::println);
```

Java 9

some nice news…

@SvenRuppert

Java9 - Optional

@SvenRuppert

Java9 - Optional

@SvenRuppert

```
final Function<Integer, Optional<Integer>> function
    = (value) -> (value <= 4) ? Optional.of(value) : Optional.empty();

final List<Integer> collectionJava8 = Stream
    .of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
    .map(function) // only to wrap
    .filter(Optional::isPresent)
    .map(Optional::get)
    .collect(Collectors.toList());

final List<Integer> collectionJava9 = Stream
    .of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
    .map(function) // only to wrap
    .flatMap(Optional::stream)
    .collect(Collectors.toList());
```

Java9 - Optional

@SvenRuppert

```
final Function<Integer, Optional<Integer>> function
    = (value) -> (value <= 4) ? Optional.of(value) : Optional.empty();

final List<Integer> collectionJava8 = Stream
    .of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
    .map(function) // only to wrap
    .filter(Optional::isPresent)
    .map(Optional::get)
    .collect(Collectors.toList());

final List<Integer> collectionJava9 = Stream
    .of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
    .map(function) // only to wrap
    .flatMap(Optional::stream)
    .collect(Collectors.toList());
```

Java9 - Optional

@SvenRuppert

```
final Function<Integer, Optional<Integer>> function
    = (value) -> (value <= 4) ? Optional.of(value) : Optional.empty();

final List<Integer> collectionJava8 = Stream
    .of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
    .map(function) // only to wrap
    .filter(Optional::isPresent)
    .map(Optional::get)
    .collect(Collectors.toList());

final List<Integer> collectionJava9 = Stream
    .of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
    .map(function) // only to wrap
    .flatMap(Optional::stream)
    .collect(Collectors.toList());
```

Java9 - Optional

@SvenRuppert

```
final Function<Integer, Optional<Integer>> function
    = (value) -> (value <= 4) ? Optional.of(value) : Optional.empty();

final List<Integer> collectionJava8 = Stream
    .of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
    .map(function) // only to wrap
    .filter(Optional::isPresent)
    .map(Optional::get)
    .collect(Collectors.toList());

final List<Integer> collectionJava9 = Stream
    .of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
    .map(function) // only to wrap
    .flatMap(Optional::stream)
    .collect(Collectors.toList());
```

Java9 - ProcessBuilder

@SvenRuppert

show code in the IDE

Summary

Now it is time to relax again ;-)

Summary

If you are interested…

have a look at **GITHUB**

ProxyBuilder

Dynamic-Dependency-Injection

Java-Microservice

or contact me ;-) @SvenRuppert

Summary

If you are interested…

have a look at **GITHUB**

Thank You !!!

ProxyBuilder

Dynamic-Dependency-Injection

Java-Microservice

or contact me ;-) @SvenRuppert