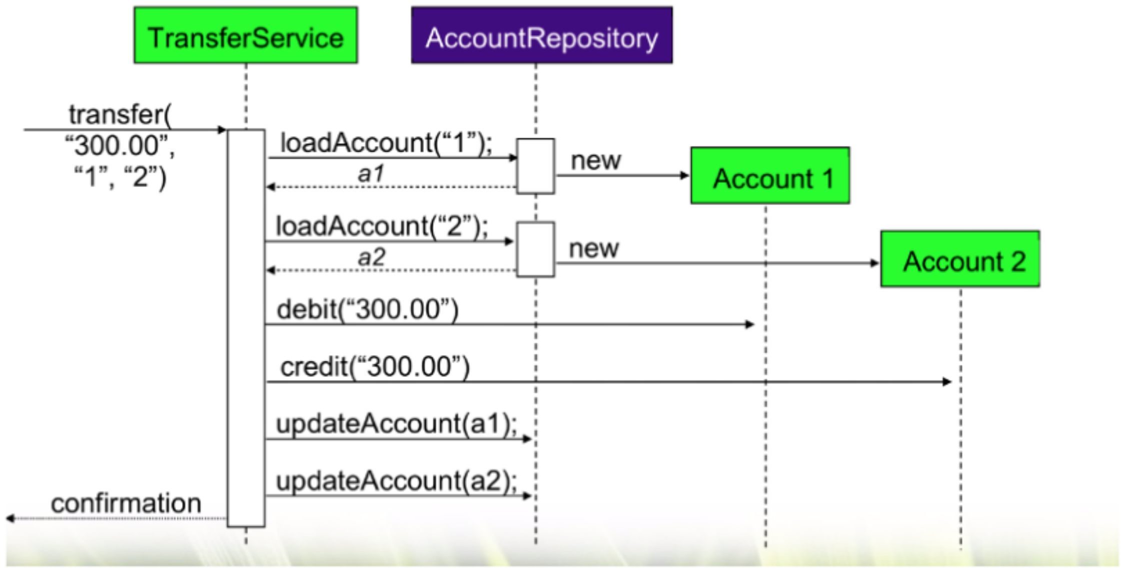
spring & spring-boot

Ex: **Money Transfer Service**



demo-1 : money-transfer-service

design & performance issues

- **tight-coupling between dependent & dependency object's implementation**

  > can't extend with new features easily

- **too many duplicate dependency instances**

  > too much resource consumption & bad responsive to end-user

- **unit-testing not possible**

  > dev / bug-fix slow

---

why these issues ?

> dependent component managing it's own dependency component

---

solution:

> don't create , do lookup on factory. **Factory Design Pattern**

---

limitation with factory only

> factory location tight-coupling

---

best solution:

> don't create & lookup , inject by container (dependency inversion principle)

S.O.L.I.D principles

1. **Single Responsibility Principle**

   > “One class should have one and only one responsibility”

2. **Open Closed Principle**

   > “Software components should be open for extension, but closed for modification”

3. **Liskov’s Substitution Principle**

   > “Derived types must be completely substitutable for their base types”

4. **Interface Segregation Principle**

> “Clients should not be forced to implement unnecessary methods which they will not use”

5. **Dependency Inversion Principle**

> “Depend on abstractions, not on concretions”

Spring configuration

1. XML
2. Annotation
3. Java-based
