```java
List<String> subjects = List.of("Noel", "The cat", "The dog");
List<String> verbs =    List.of("wrote", "chased", "slept on");
List<String> objects =  List.of("the book","the ball","the bed");

for (String s: subjects) {
  for (String v: verbs) {
    for (String o: objects) {
      System.out.printf("%s %s %s\n", s, v, o);
    }
  }
}
```

the thing is done

no way to reuse this functionality !!!

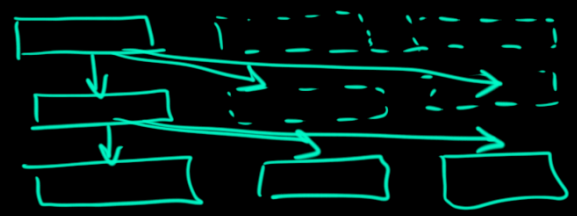
```java
ArrayList<String> sentences = new ArrayList<>();
for (String s: subjects) {
  for (String v: verbs) {
    for (String o: objects) {
      String sentence = String.format("%s %s %s", s, v, o);
      sentences.add(sentence);
    }
  }
}

sentences.forEach(System.out::println);
```

1. we are creating structure

2. we can reuse this functionality

1)

```
ArrayList<Sentence> sentences = new ArrayList<>();
for (String s: subjects) {
  for (String v: verbs) {
    for (String o: objects) {
      sentences.add(new Sentence(s, o, v));
    }
  }
}
return sentences;
```




```
Stream<Stream<Stream<Sentence>>> sentences = subjects.stream().map(s ->
    verbs.stream().map(v ->
        objects.stream().map(o ->
            new Sentence(s, v, o)
        )
    )
);
```

.map (A -> B)

Stream<A> .map (A->B) | Stream<B>

A -> List<B> | Stream< List<B> >

5)

```
return subjects.stream().flatMap(s ->
    verbs.stream().flatMap(v ->
        objects.stream().map(o ->
            new Sentence(s, v, o)
        )
    )
```

.flatMap (A -> Stream<B>)

(2) (1)

.flatMap (A -> Stream<B>)
.map (A -> Stream<B>) .flatten
Stream< Stream<A> > -> Stream<A>

```
List.of(1,2,3)
```

→ List(1,1,2,2,3,3)

```
.flatMap(x -> Stream.of(x, x))
```

↘ List(1,-1,2,-2,3,-3)

```
.flatMap(x -> Stream.of(x, -x))
```

DAO

pattern

Data Access Object

```java
public class DaoPersonFile {

  private final File file;

  public DaoPersonFile(File file) {
    this.file = file;
  }

  public void save(Person p) {
    throw new IllegalArgumentException("not implemented");
  }

  public Person load(int id) {
    throw new IllegalArgumentException("not implemented");
  }

}
```

```java
public class DaoPersonInMemory {

  private final Map<Integer, Person> map = new HashMap<>();

  public void save(Person p) {
    throw new IllegalArgumentException("not implemented");
  }

  public Person load(int id) {
    throw new IllegalArgumentException("not implemented");
  }

}
```

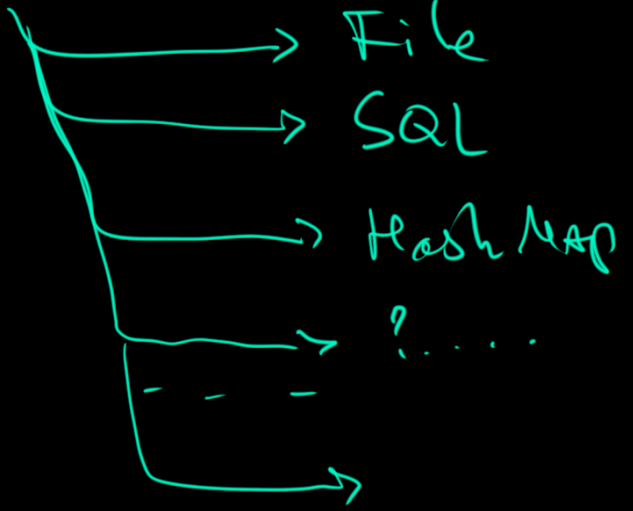
empty constr.

```java
public class DaoPersonSQLDb {

  private final Connection sqlConnection;

  public DaoPersonSQLDb(Connection sqlConnection) {
    this.sqlConnection = sqlConnection;
  }

  public void save(Person p) {
    throw new IllegalArgumentException("not implemented");
  }

  public Person load(int id) {
    throw new IllegalArgumentException("not implemented");
  }

}
```

they have different constructors

Interface

same



if I need to switch implementation => I need to modify that

many ...

```java
public void soSomething(DaoPersonFile daoPersonFile) {
  /// ....
}
```

```java
public interface DaoPerson {
  void save(Person p);
  Person load(int id);
}
```

interface

```java
public void soSomething(DaoPerson daoPerson) {
  /// ....
}
```

do we need to be aware of the another / diff impl

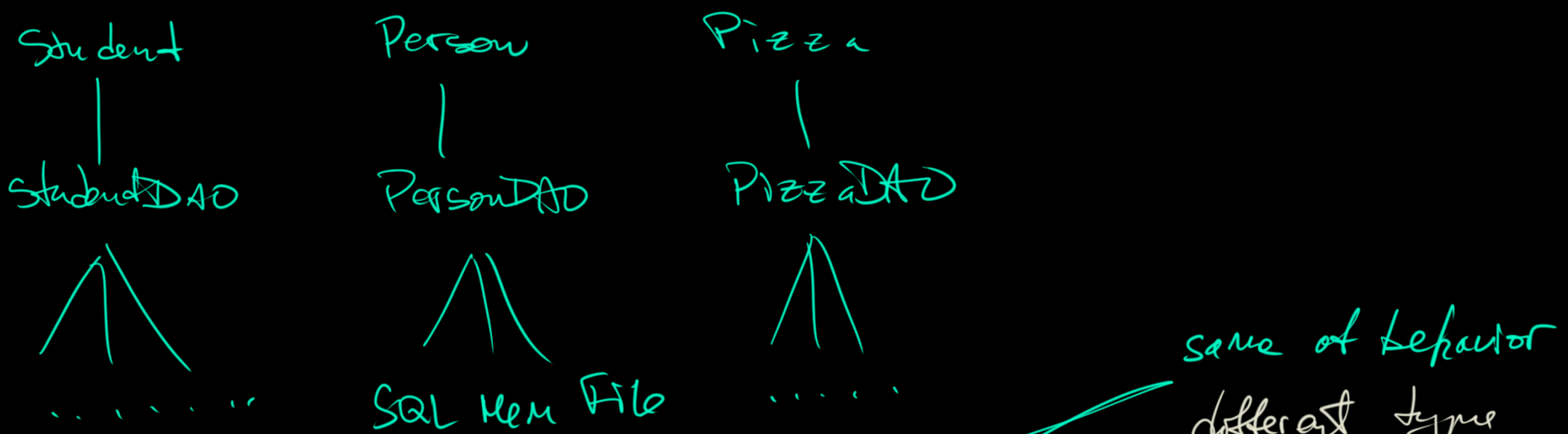
= NO

```java
public class DaoPersonSQLDb implements DaoPerson {

  private final Connection sqlConnection;

  public DaoPersonSQLDb(Connection sqlConnection) {
    this.sqlConnection = sqlConnection;
  }

  @Override
  public void save(Person p) {
    throw new IllegalArgumentException("not implemented");
  }

  @Override
  public Person load(int id) {
    throw new IllegalArgumentException("not implemented");
  }
```

```java
public class DaoPersonInMemory implements DaoPerson {
```

```java
public class DaoPersonFile implements DaoPerson {
```

```java
DaoPerson daoPersonInmemory = new DaoPersonInMemory();
daoPersonInmemory.save(p);
Person p1 = daoPersonInmemory.load(id:123);

DaoPerson daoPersonFile = new DaoPersonFile(new File(pa
daoPersonFile.save(p);
Person p2 = daoPersonFile.load(id:123);

soSomething(daoPersonInmemory);
soSomething(daoPersonFile);
```




```
public interface DaoPizza {
  void save(Pizza p);
  Pizza load(int id);
}
```

```
public interface DaoStudent {
  void save(Student p);
  Student load(int id);
}
```

A B

T U V W X Y Z (7)

A B C D E F G H . . . Z (26)

```
public interface DAO<A> {
    4 implementations
    void save(A p);
    4 implementations
    A load(int id);
}
```

A=Pizza

```
public interface DaoPizza extends DAO<Pizza> {
}
```

```
public interface DaoPerson extends DAO<Person>{
}
```

A=Person

```
public class DaoPersonFile implements DaoPerson {

    private final File file;

    public DaoPersonFile(File file) { this.file = file; }

    @Override
    public void save(Person p) { throw new IllegalArgumentExce

    @Override
    public Person load(int id) { throw new IllegalArgumentExce

}
```

DAO<Person>

```
public class DaoPersonInMemory implements DaoPerson {

    private final Map<Integer, Person> map = new HashMap<>(

    @Override
    public void save(Person p) { throw new IllegalArgumentE

    @Override
    public Person load(int id) { throw new IllegalArgumentE

}
```

```java
DaoPizzaInMemory daoPizzaInMemory1 = new DaoPizzaInMemory();
DaoPizza         daoPizzaInMemory2 = new DaoPizzaInMemory();
DAO<Pizza>       daoPizzaInMemory3 = new DaoPizzaInMemory();
```

DaoPizzaInMemory ≡ DaoPizza ≡ DAO<Pizza>