Inheritance: Table per Class

Inheritance Mapping Strategies

- Single table
- **Table per class**
- Joined table
- Mapped superclass

Inheritance Mapping Strategies

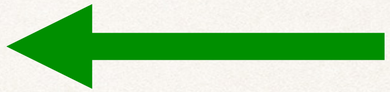
- Single table
- **Table per class** 
- Joined table
- Mapped superclass

Table per Class

Table per Class

- For the inheritance tree, all concrete classes are mapped **a table per class**

Table per Class

- For the inheritance tree, all concrete classes are mapped **a table per class**
  - Remember, a concrete class is a non-abstract class

Table per Class

- For the inheritance tree, all concrete classes are mapped **a table per class**
  - Remember, a concrete class is a non-abstract class

- Includes fields defined in subclass and fields inherited from superclass

Table per Class

- For the inheritance tree, all concrete classes are mapped **a table per class**
  - Remember, a concrete class is a non-abstract class
- Includes fields defined in subclass and fields inherited from superclass
- These fields are mapped to a table for the concrete class

Table per Class

Table per Class

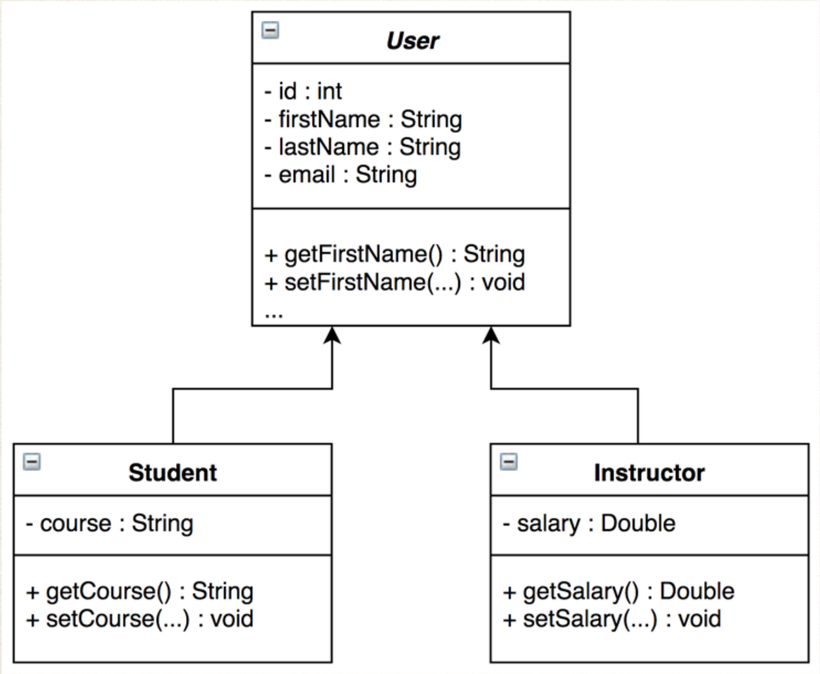


Table per Class

abstract

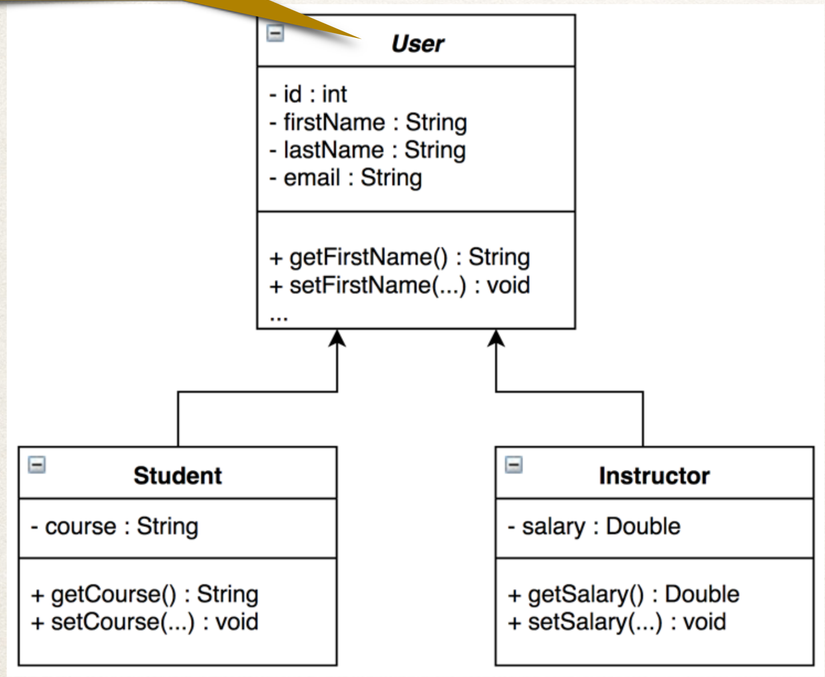


Table per Class

abstract

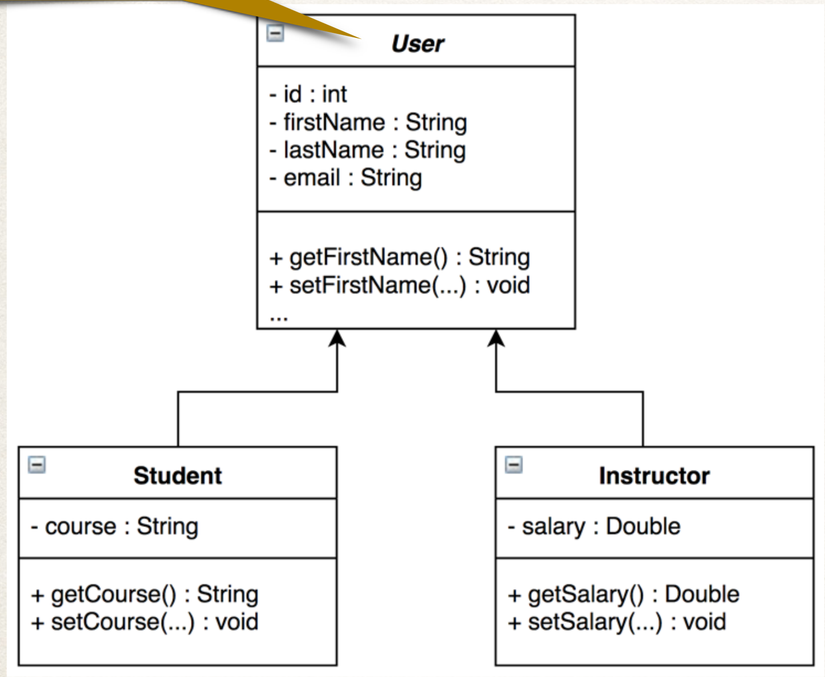
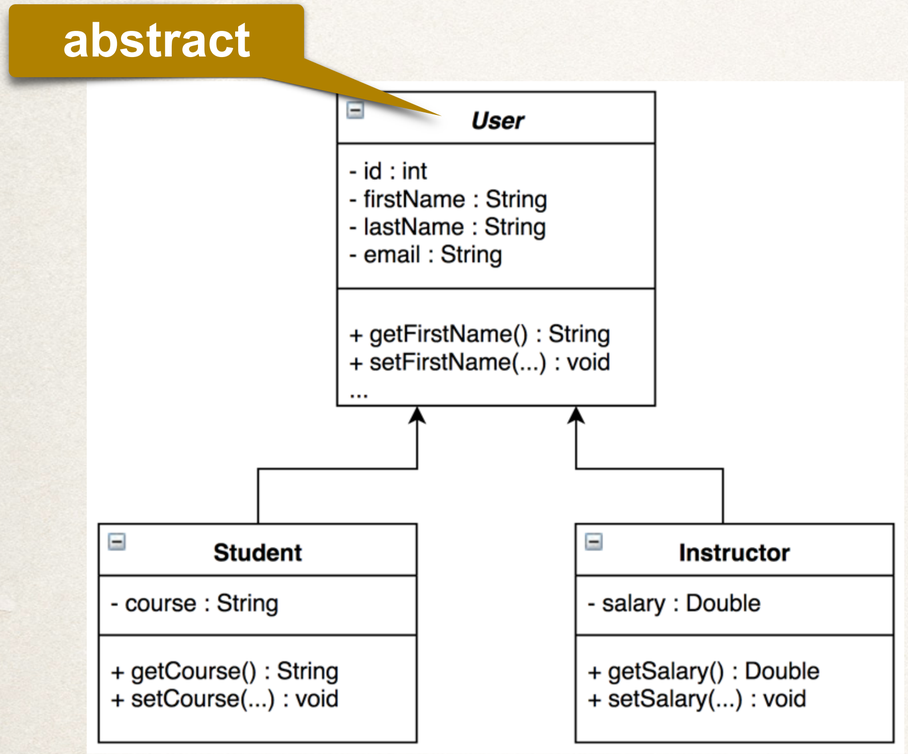
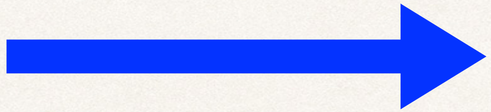


Table per Class





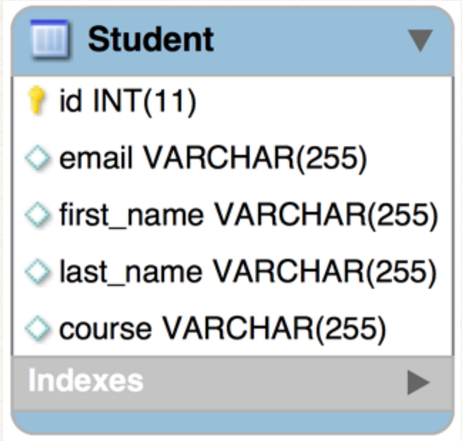
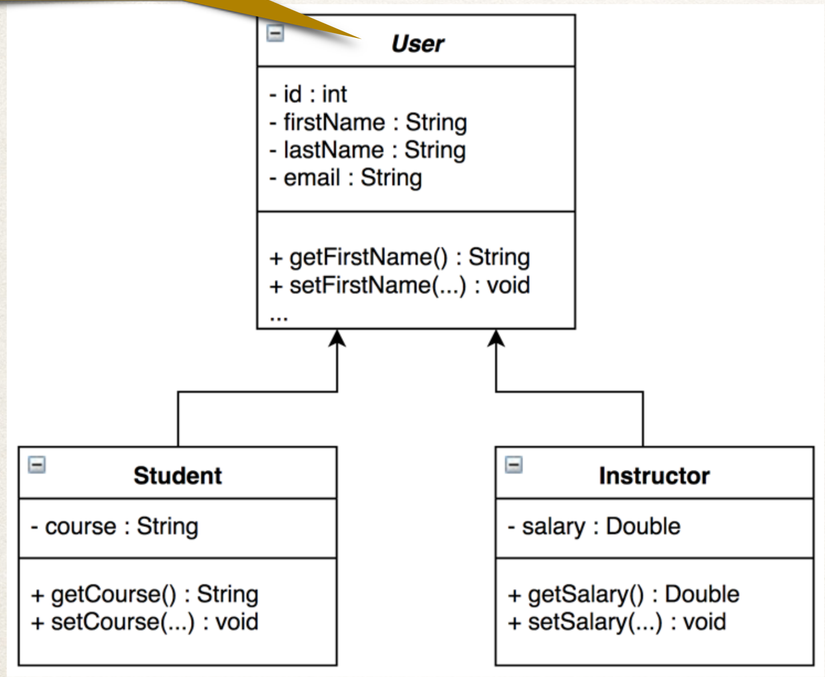
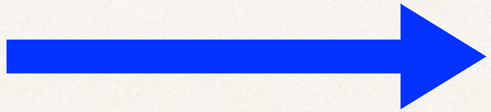
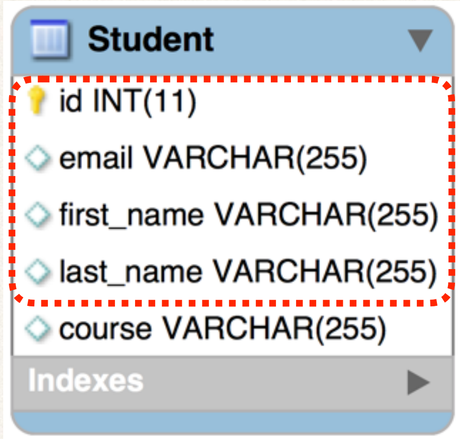


Table per Class

abstract



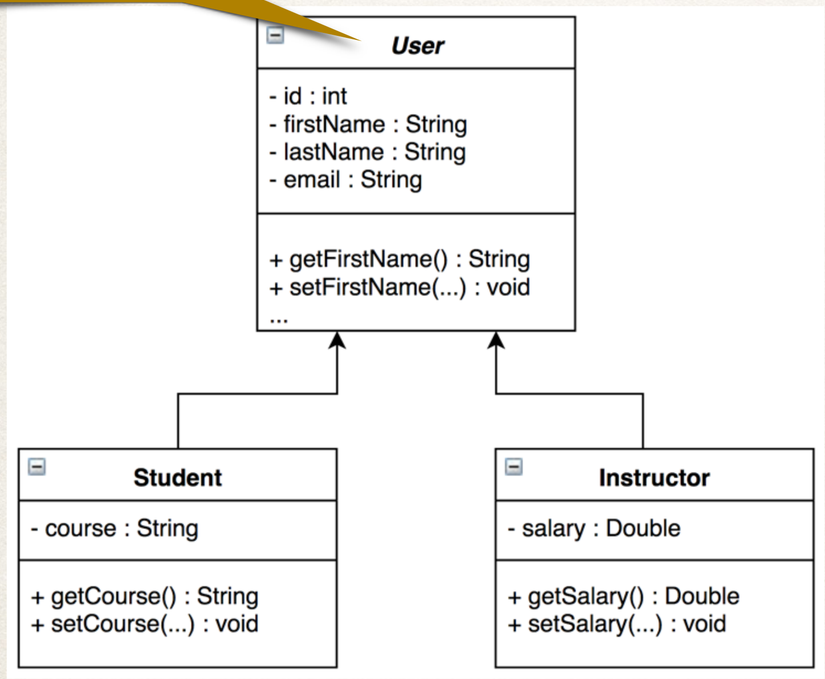


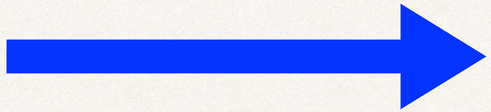


User

Table per Class

abstract





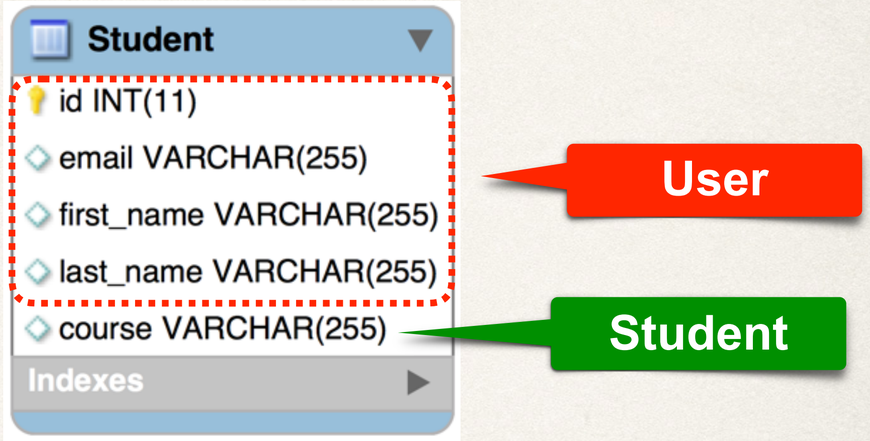


Table per Class

abstract

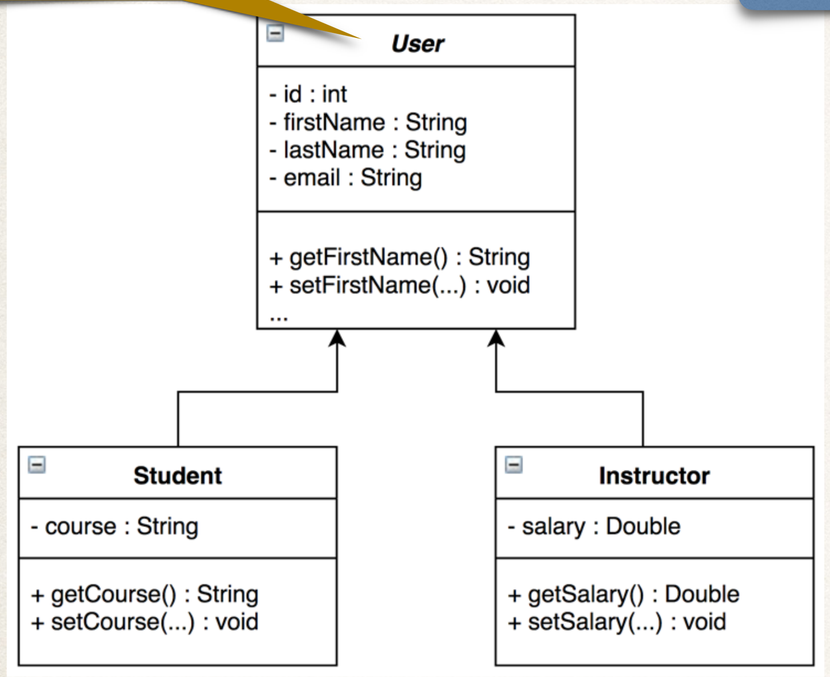
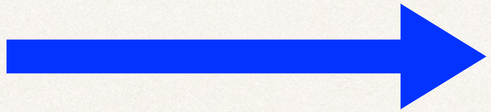
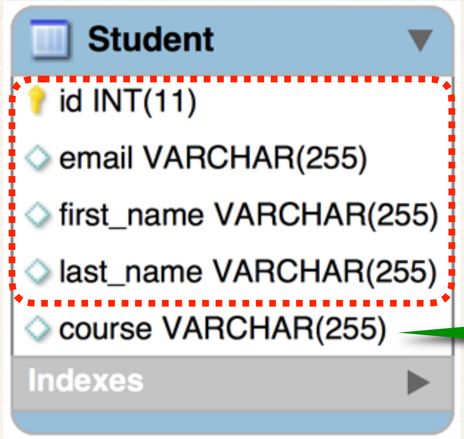


Table per class for the concrete classes





User

Student

Table per Class

abstract

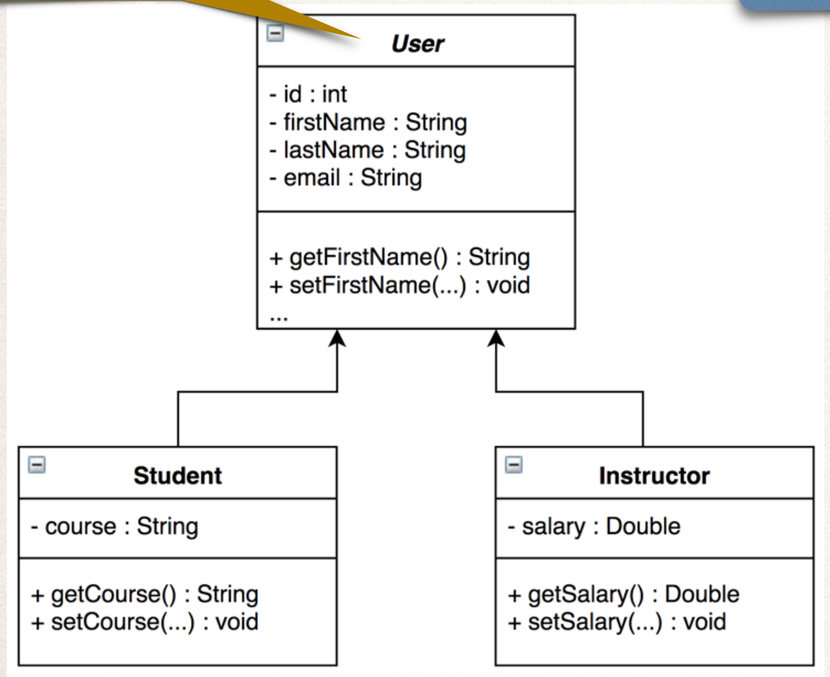
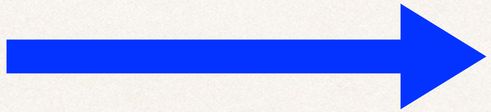
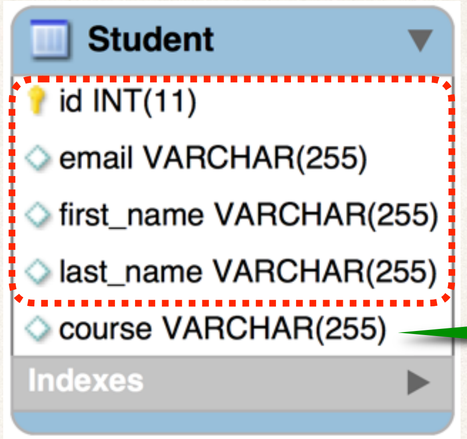


Table per class for the concrete classes





User

Student

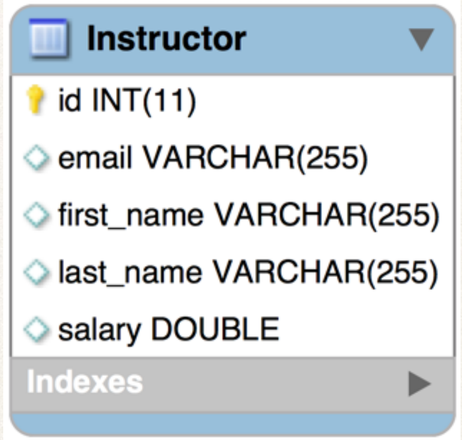


Table per Class

abstract

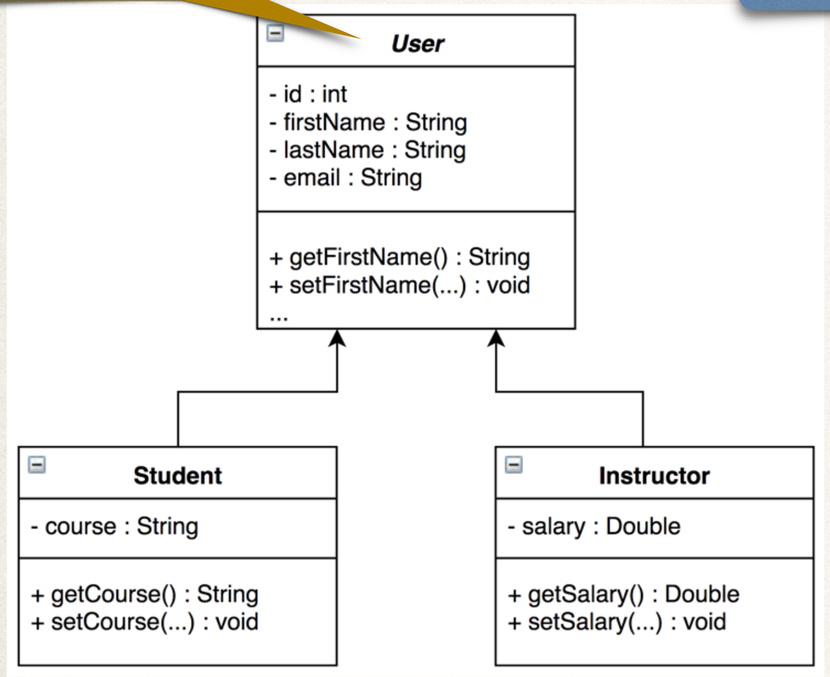
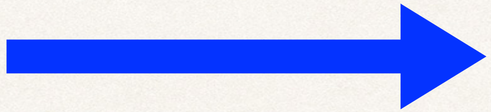
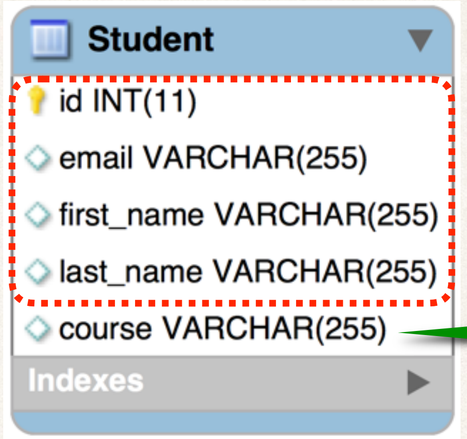


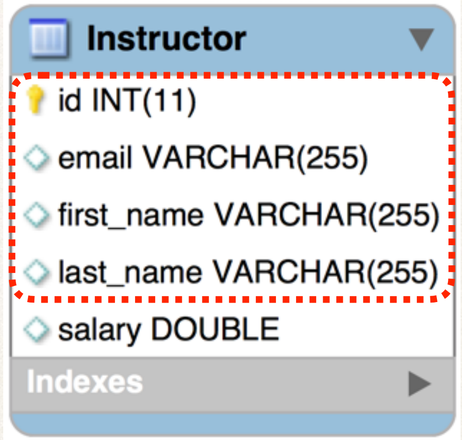
Table per class for the concrete classes





User

Student



User

Table per Class

abstract

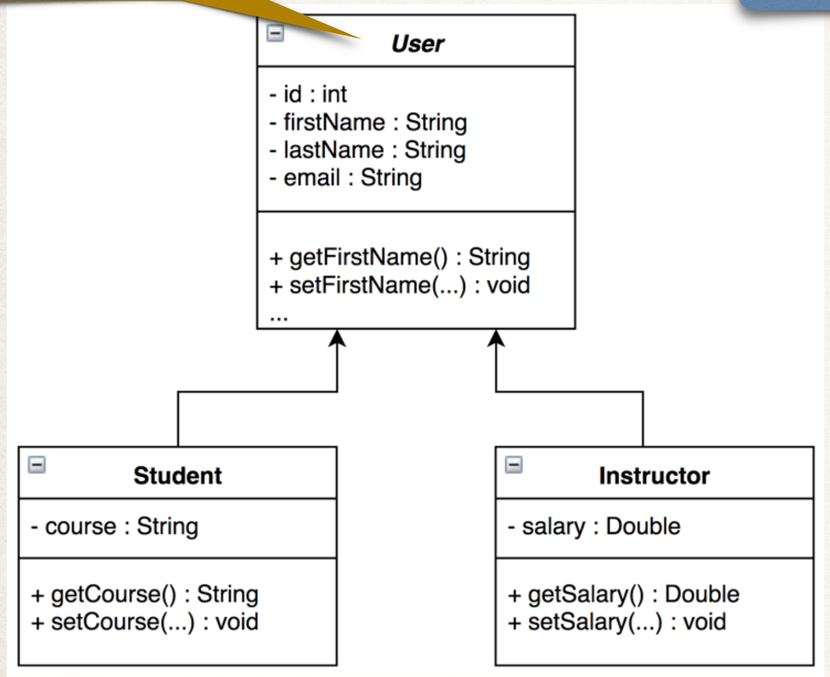


Table per class for the concrete classes

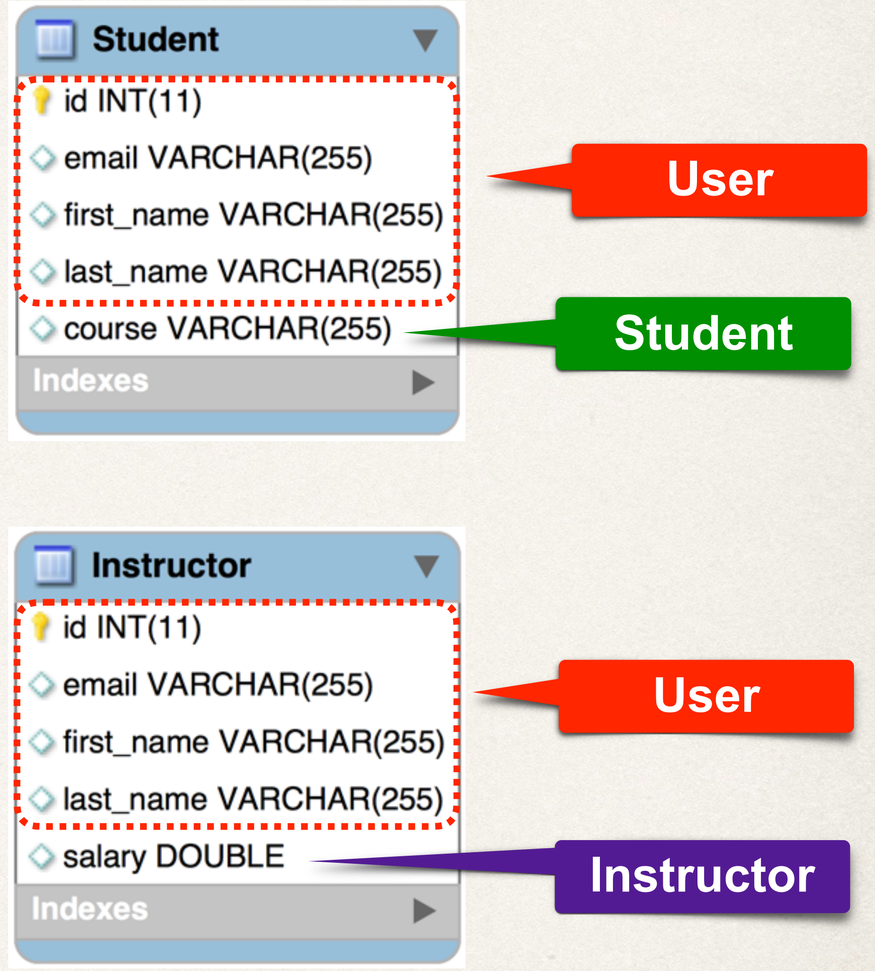


Table per Class

abstract

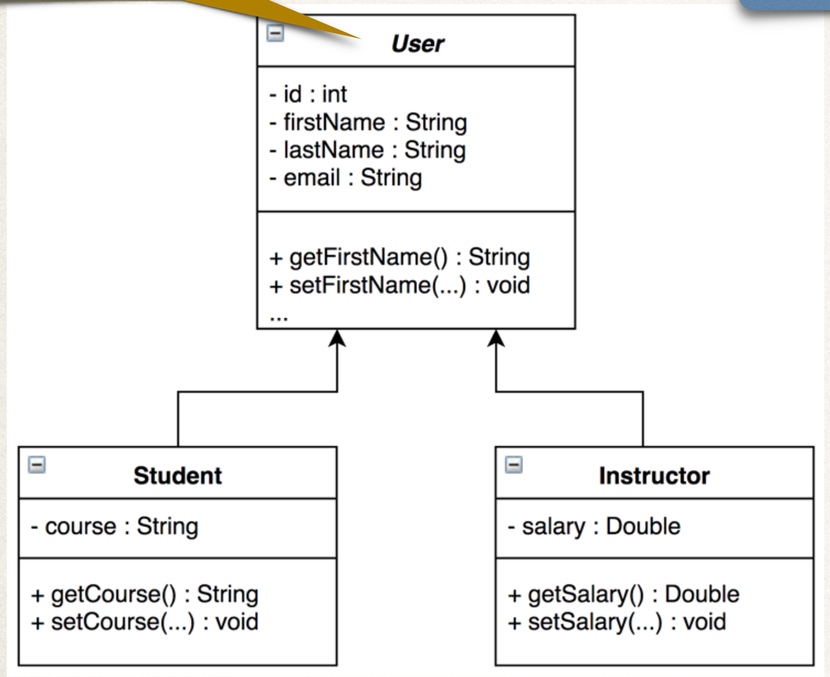
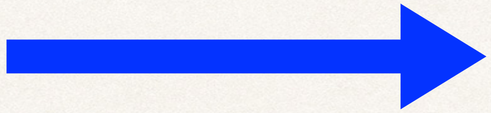
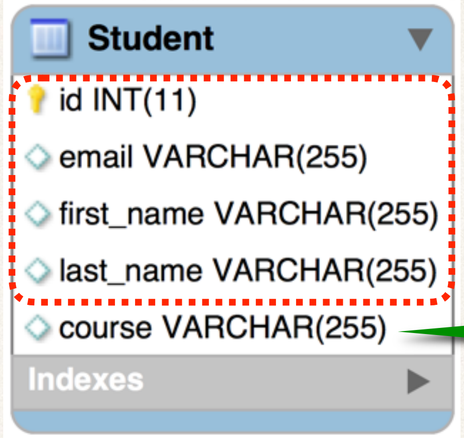
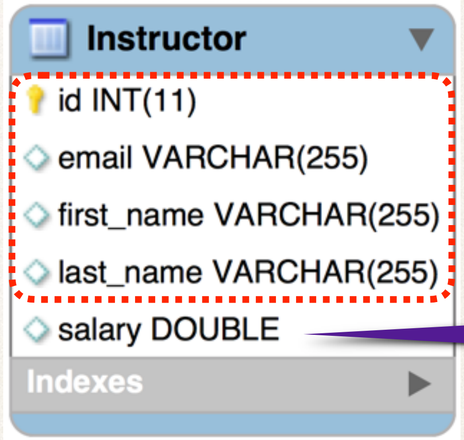


Table per class for the concrete classes



Note: Only the related fields
No extra columns





Development Process

Step-By-Step

Development Process

Step-By-Step

1. In superclass, specify inheritance strategy: `TABLE_PER_CLASS`

Development Process

Step-By-Step

1. In superclass, specify inheritance strategy: `TABLE_PER_CLASS`

2. Remove annotations for Discriminator

Development Process

Step-By-Step

1. In superclass, specify inheritance strategy: `TABLE_PER_CLASS`

2. Remove annotations for Discriminator

3. In `User` superclass, update the ID generation strategy to `TABLE`

Development Process

Step-By-Step

1. In superclass, specify inheritance strategy: `TABLE_PER_CLASS`

2. Remove annotations for Discriminator

3. In `User` superclass, update the ID generation strategy to `TABLE`

4. Make updates to Hibernate configuration file

Development Process

1. In superclass, specify inheritance strategy: `TABLE_PER_CLASS`

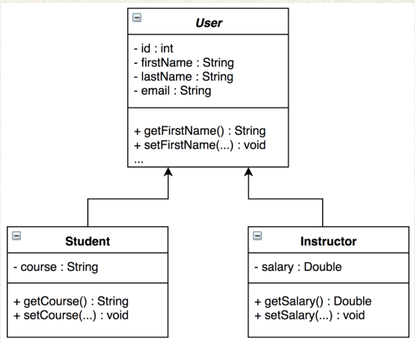
2. Remove annotations for Discriminator

3. In `User` superclass, update the ID generation strategy to `TABLE`

4. Make updates to Hibernate configuration file

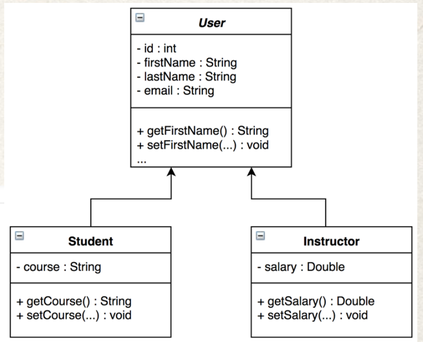
5. Develop main application

Step 1: Superclass - Inheritance strategy



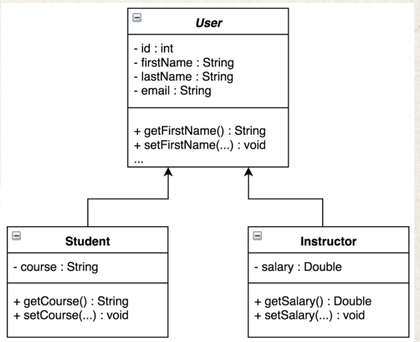
Step 1: Superclass - Inheritance strategy

```
@Entity
@Table(name="user")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class User {

...

}
```



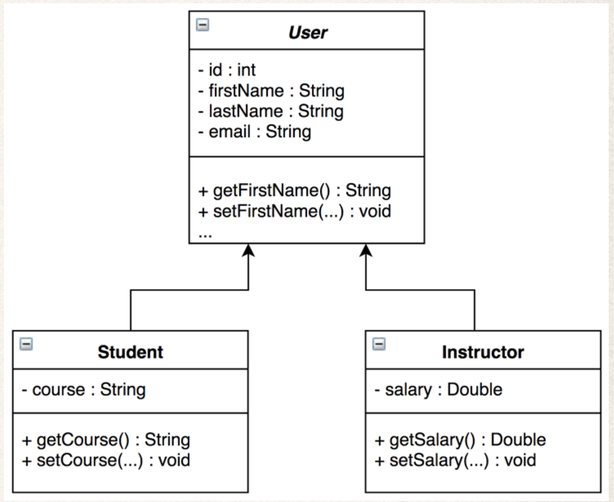
Step 1: Superclass - Inheritance strategy

```
@Entity
@Table(name="user")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class User {

...

}
```



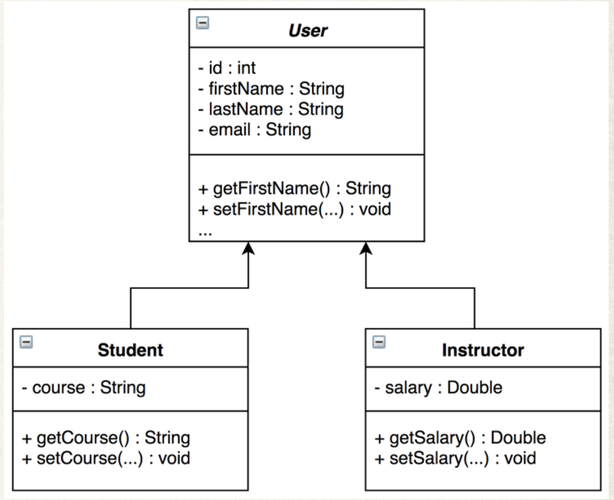
Map fields to table per class (concrete classes)

Step 2: Remove annotations for Discriminator



Step 2: Remove annotations for Discriminator

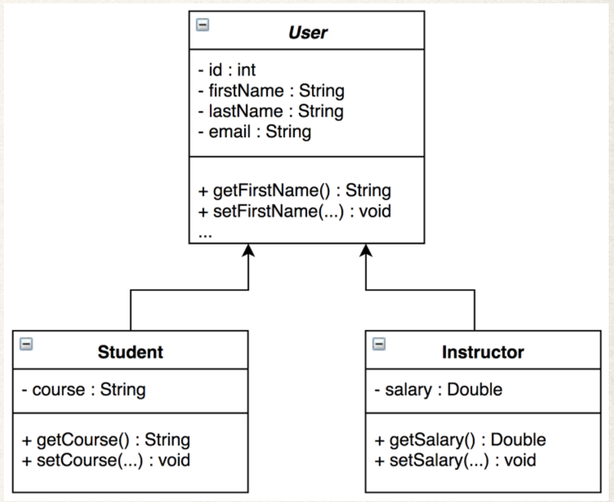
```
@Entity
@Table(name="user")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class User {
  ...
}
```



Step 2: Remove annotations for Discriminator

```
@Entity
@Table(name="user")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class User {
  ...
}
```

No need for discriminator columns or values

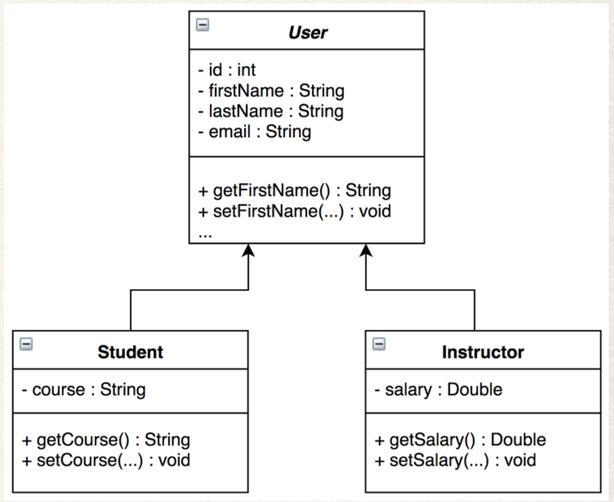


Step 2: Remove annotations for Discriminator

```
@Entity
@Table(name="user")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class User {
  ...
}
```

```
@Entity
public class Student extends User {
  ...
}
```

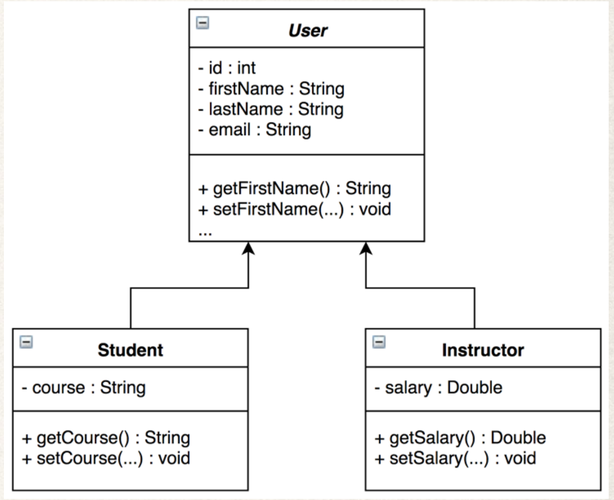
No need for discriminator columns or values



Step 2: Remove annotations for Discriminator

```
@Entity
@Table(name="user")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class User {
  ...
}
```

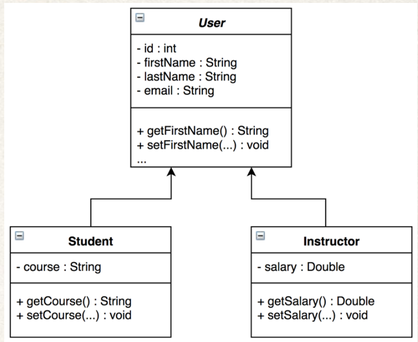
No need for discriminator columns or values




```
@Entity
public class Student extends User {
  ...
}
```

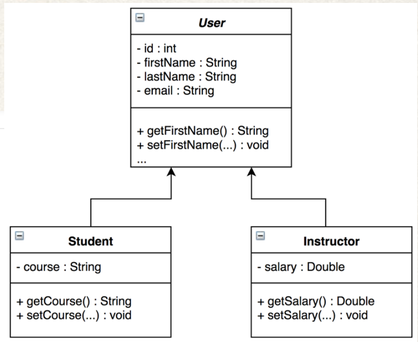
```
@Entity
public class Instructor extends User {
  ...
}
```

Step 3: Update ID generation strategy



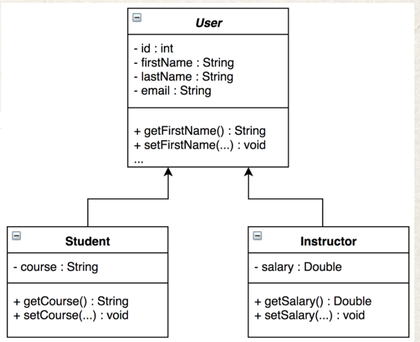
Step 3: Update ID generation strategy

```
@Entity
@Table(name="user")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class User {

  @Id
  @GeneratedValue(strategy=GenerationType.TABLE)
  @Column(name="id")
  private int id;

}
```



Step 3: Update ID generation strategy

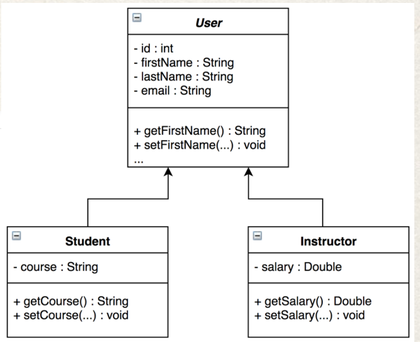
```
@Entity
@Table(name="user")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class User {

  @Id
  @GeneratedValue(strategy=GenerationType.TABLE)
  @Column(name="id")
  private int id;

}
```



When using TABLE_PER_CLASS strategy, it is required to use TABLE generation strategy

Step 3: Update ID generation strategy

```
@Entity
@Table(name="user")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public abstract class User {

  @Id
  @GeneratedValue(strategy=GenerationType.TABLE)
  @Column(name="id")
  private int id;

}
```



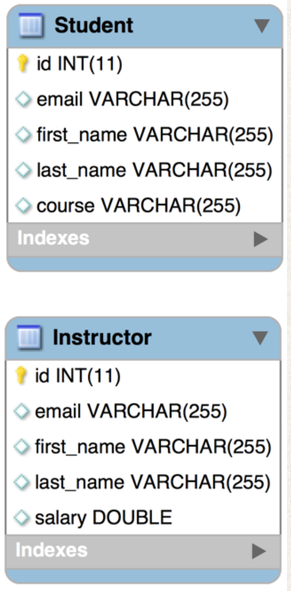
When using TABLE_PER_CLASS strategy, it is required to use TABLE generation strategy

Since entities span multiple tables, the next generated ID value can be maintained in a separate table

Step 3: Update ID generation strategy

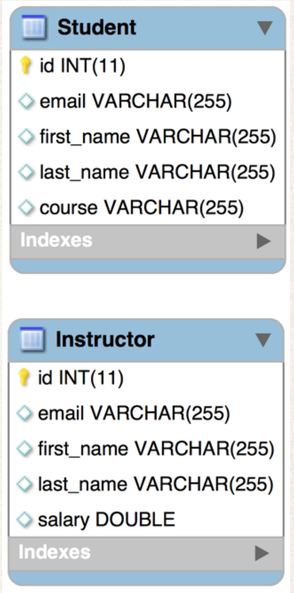
Step 3: Update ID generation strategy

- Each table has an ID field



Step 3: Update ID generation strategy

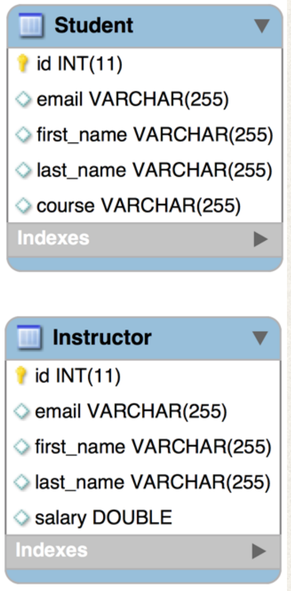
- Each table has an ID field



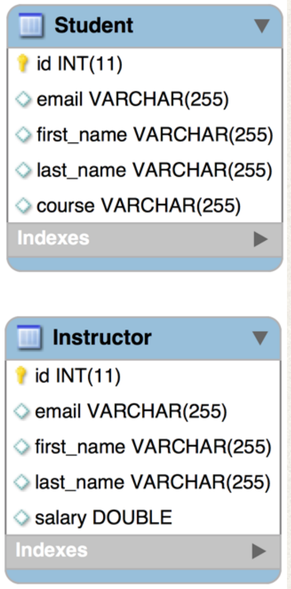
Step 3: Update ID generation strategy

- Each table has an ID field

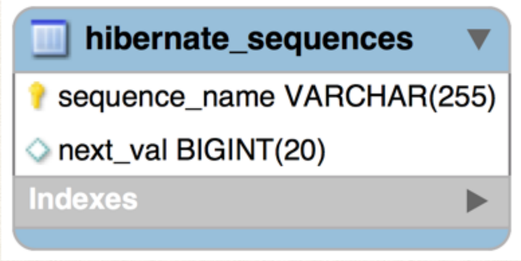
- When we add an entity, we need the next generated ID



Step 3: Update ID generation strategy



- Each table has an ID field
- When we add an entity, we need the next generated ID
- The next generated ID value can be maintained in a separate table
- Known as a “sequence table”



Step 3: Update ID generation strategy

Step 3: Update ID generation strategy

- When using Hibernate DDL auto create feature

Step 3: Update ID generation strategy

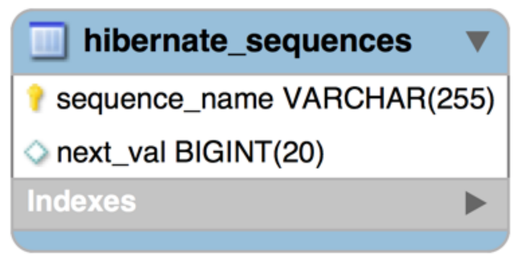
- When using Hibernate DDL auto create feature
- Hibernate will create a sequence table automatically

Step 3: Update ID generation strategy

- When using Hibernate DDL auto create feature
- Hibernate will create a sequence table automatically
- This sequence table will hold the “next_val” to use as a base for next id

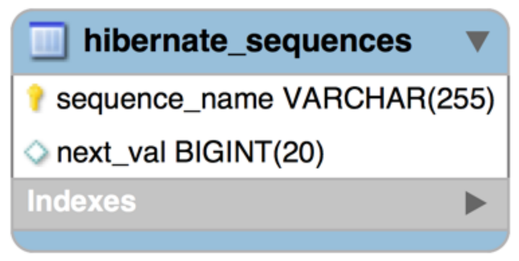
Step 3: Update ID generation strategy

- When using Hibernate DDL auto create feature
- Hibernate will create a sequence table automatically
- This sequence table will hold the “next_val” to use as a base for next id



Step 3: Update ID generation strategy

- When using Hibernate DDL auto create feature
- Hibernate will create a sequence table automatically
- This sequence table will hold the “next_val” to use as a base for next id

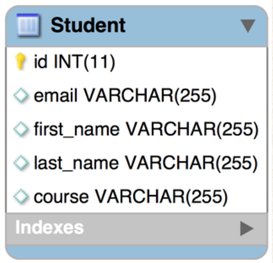


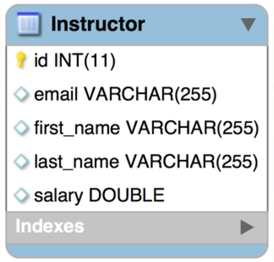
**Behind the scenes,
Hibernate will get the next value
from the sequence table**

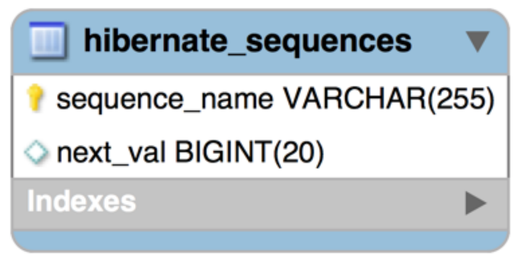
**and then increment it by 1
(thread-safe)**

Step 3: Update ID generation strategy

- When using Hibernate DDL auto create feature
- Hibernate will create a sequence table automatically
- This sequence table will hold the "next_val" to use as a base for next id





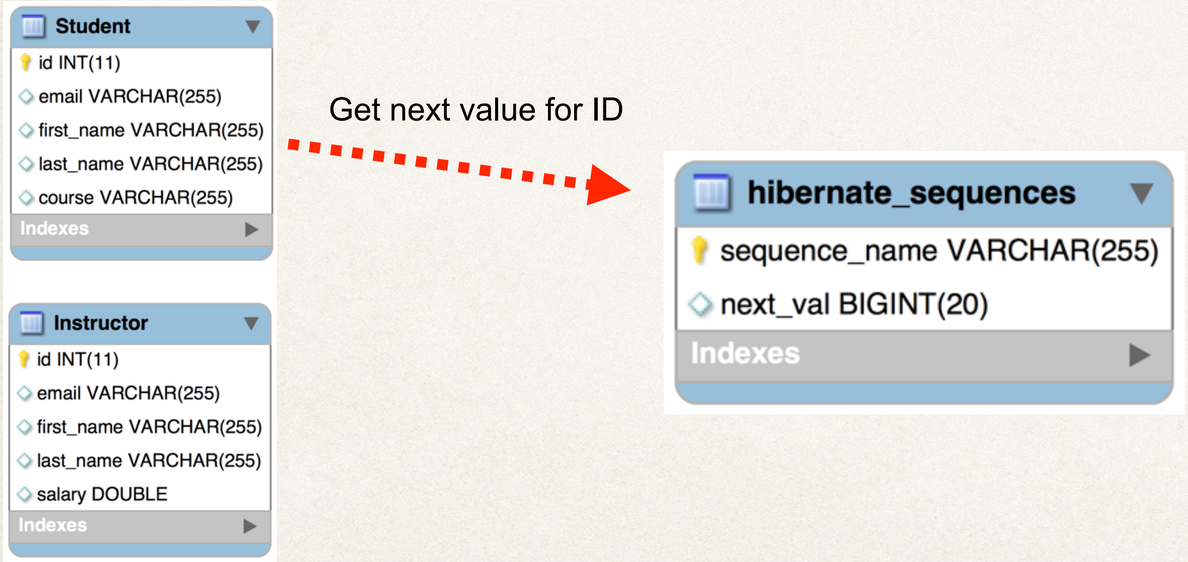


Behind the scenes, Hibernate will get the next value from the sequence table

and then increment it by 1 (thread-safe)

Step 3: Update ID generation strategy

- When using Hibernate DDL auto create feature
- Hibernate will create a sequence table automatically
- This sequence table will hold the “next_val” to use as a base for next id

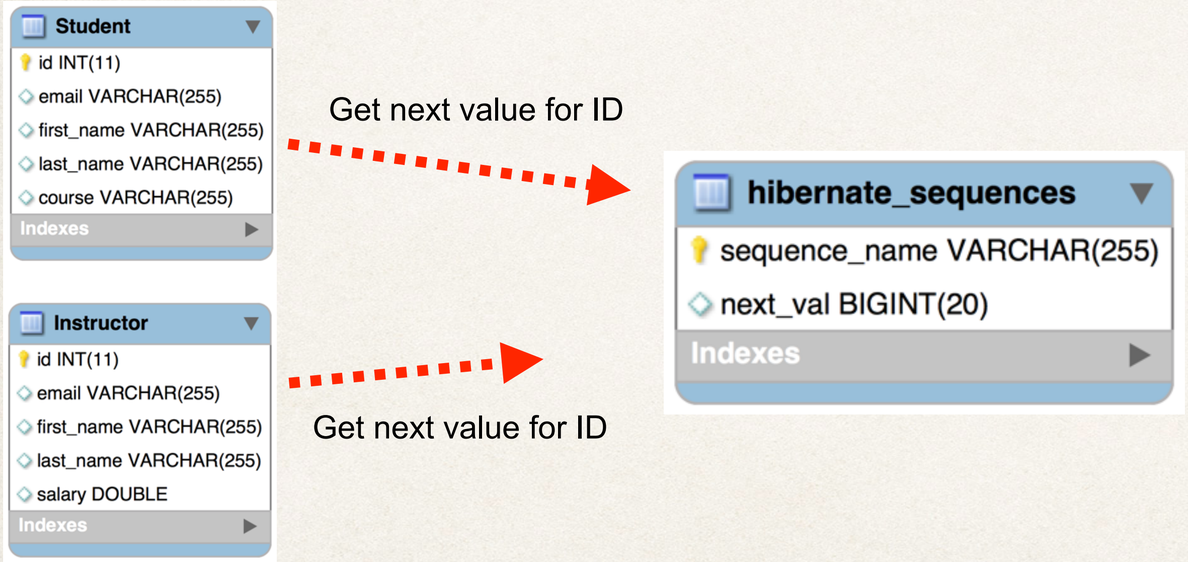


Behind the scenes, Hibernate will get the next value from the sequence table

and then increment it by 1 (thread-safe)

Step 3: Update ID generation strategy

- When using Hibernate DDL auto create feature
- Hibernate will create a sequence table automatically
- This sequence table will hold the "next_val" to use as a base for next id

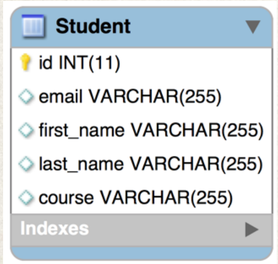


Behind the scenes, Hibernate will get the next value from the sequence table

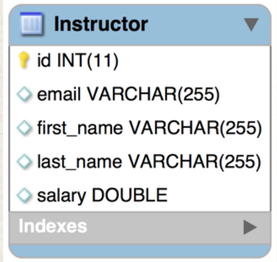
and then increment it by 1 (thread-safe)

Step 3: Update ID generation strategy

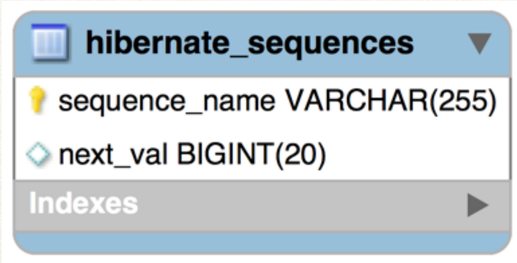
- When using Hibernate DDL auto create feature
- Hibernate will create a sequence table automatically
- This sequence table will hold the “next_val” to use as a base for next id



Get next value for ID



Get next value for ID




```
@Id
@GeneratedValue(strategy=GenerationType.TABLE)
@Column(name="id")
private int id;
```

Behind the scenes, Hibernate will get the next value from the sequence table

and then increment it by 1 (thread-safe)

Step 4: Update Hibernate Config file

Step 4: Update Hibernate Config file

```
<!-- JDBC connection pool settings ... using built-in test pool -->
<property name="connection.pool_size">10</property>

<!-- Select our SQL dialect -->
<property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>
```

Step 4: Update Hibernate Config file

Increase pool size since additional connections are needed for accessing sequence table

```
<!-- JDBC connection pool settings ... using built-in test pool -->
<property name="connection.pool_size">10</property>

<!-- Select our SQL dialect -->
<property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>
```

Step 4: Update Hibernate Config file

Increase pool size since additional connections are needed for accessing sequence table

```
<!-- JDBC connection pool settings ... using built-in test pool -->
<property name="connection.pool_size">10</property>

<!-- Select our SQL dialect -->
<property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>
```

Upgrade to MySQL 8 dialect since we are using MySQL 8 database

Needed for creation of sequence table

Step 5: Develop the main application

Step 5: Develop the main application

Student's course

```
// create the objects
Student tempStudent = new Student("Mary", "Public", "mary@luv2code.com", "Hibernate");
Instructor tempInstructor = new Instructor("John", "Doe", "john@luv2code.com", 20000.00);
```

Instructor's salary

Step 5: Develop the main application

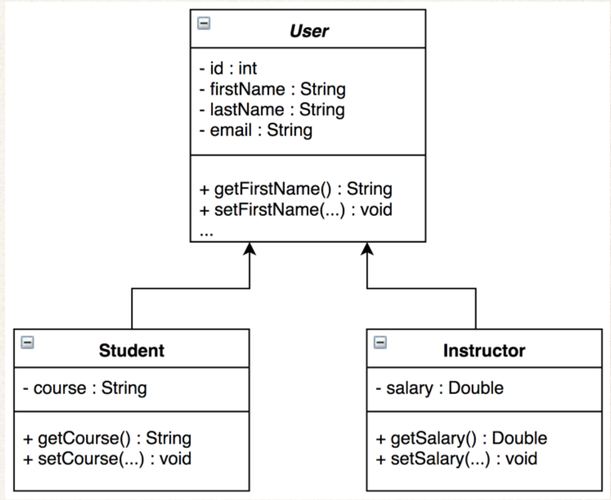
Student's course

```
// create the objects
Student tempStudent = new Student("Mary", "Public", "mary@luv2code.com", "Hibernate");
Instructor tempInstructor = new Instructor("John", "Doe", "john@luv2code.com", 20000.00);

// start a transaction
session.beginTransaction();

// save the objects
System.out.println("Saving the student and instructor...");
session.save(tempStudent);
session.save(tempInstructor);

// commit the transaction
session.getTransaction().commit();
```

Instructor's salary

All of this code is the same

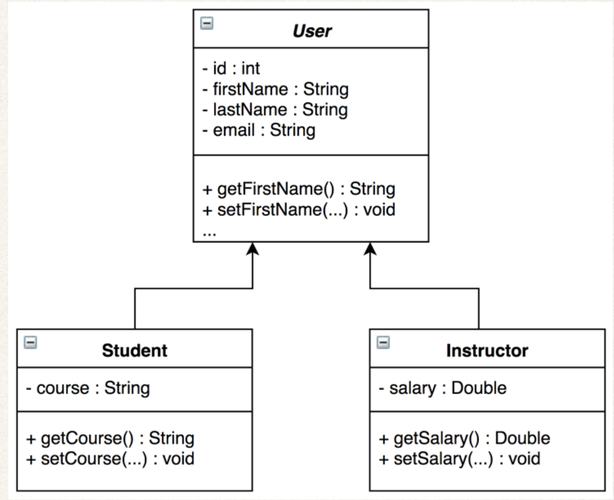
Run the App



Run the App

Console

```
Hibernate: insert into Student (email, first_name, last_name, course, id) values (?, ?, ?, ?, ?)
Hibernate: insert into Instructor (email, first_name, last_name, salary, id) values (?, ?, ?, ?, ?)
```



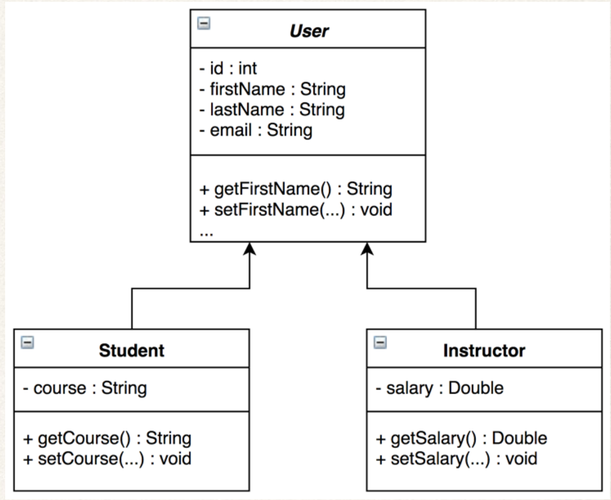
Run the App

Console

```
Hibernate: insert into Student (email, first_name, last_name, course, id) values (?, ?, ?, ?, ?)
Hibernate: insert into Instructor (email, first_name, last_name, salary, id) values (?, ?, ?, ?, ?)
```

Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate



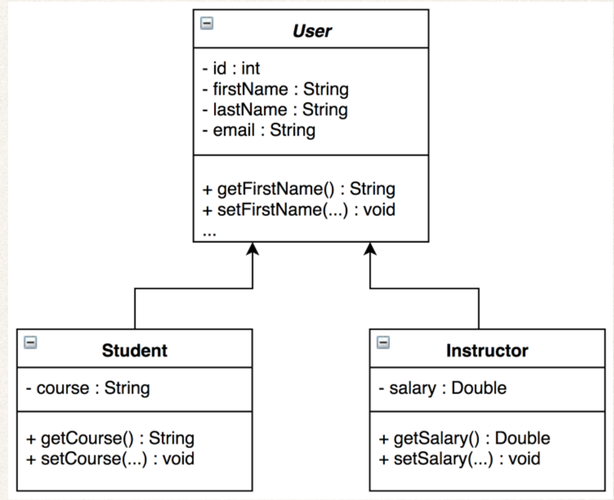
Run the App

Console

```
Hibernate: insert into Student (email, first_name, last_name, course, id) values (?, ?, ?, ?, ?)
Hibernate: insert into Instructor (email, first_name, last_name, salary, id) values (?, ?, ?, ?, ?)
```

Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate



Includes fields defined in subclass and fields inherited from superclass

Run the App

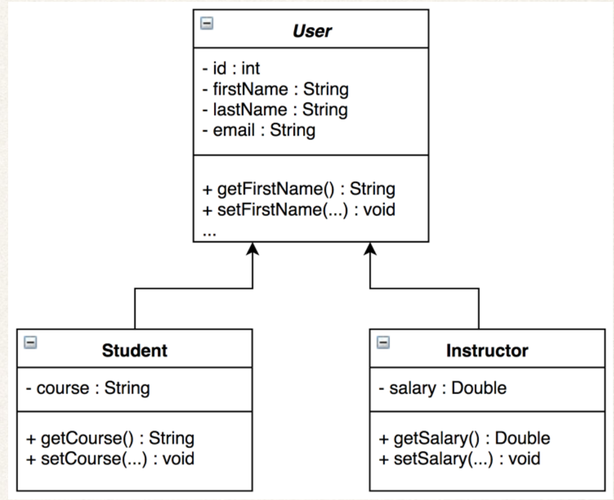
Console

```
Hibernate: insert into Student (email, first_name, last_name, course, id) values (?, ?, ?, ?, ?)
Hibernate: insert into Instructor (email, first_name, last_name, salary, id) values (?, ?, ?, ?, ?)
```

Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate

User



Includes fields defined in subclass and fields inherited from superclass

Run the App

Console

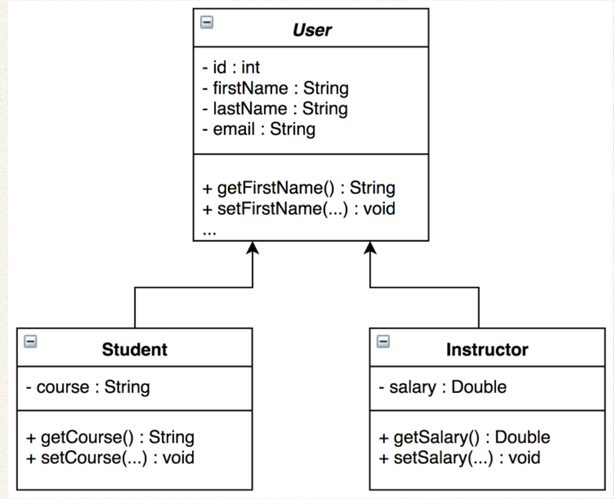
```
Hibernate: insert into Student (email, first_name, last_name, course, id) values (?, ?, ?, ?, ?)
Hibernate: insert into Instructor (email, first_name, last_name, salary, id) values (?, ?, ?, ?, ?)
```

Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate

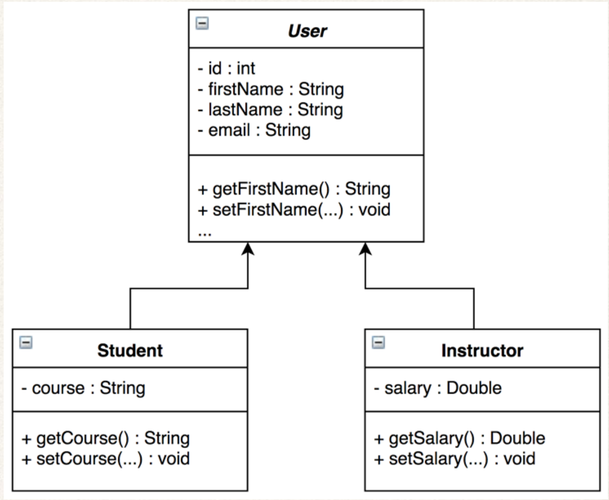
User

Student



Includes fields defined in subclass and fields inherited from superclass

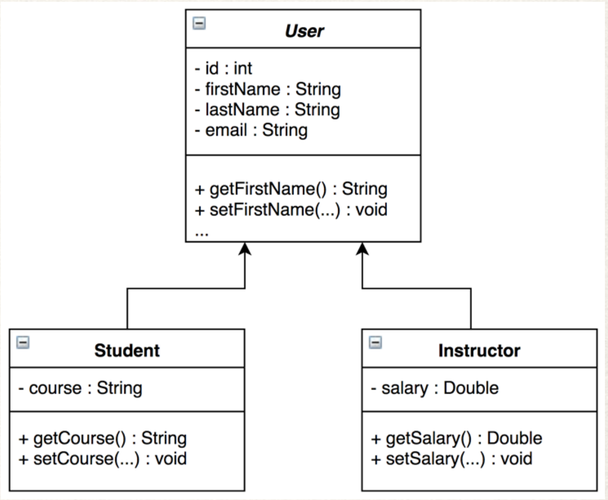
Run the App



Run the App

Console

```
Hibernate: insert into Student (email, first_name, last_name, course, id) values (?, ?, ?, ?, ?)
Hibernate: insert into Instructor (email, first_name, last_name, salary, id) values (?, ?, ?, ?, ?)
```



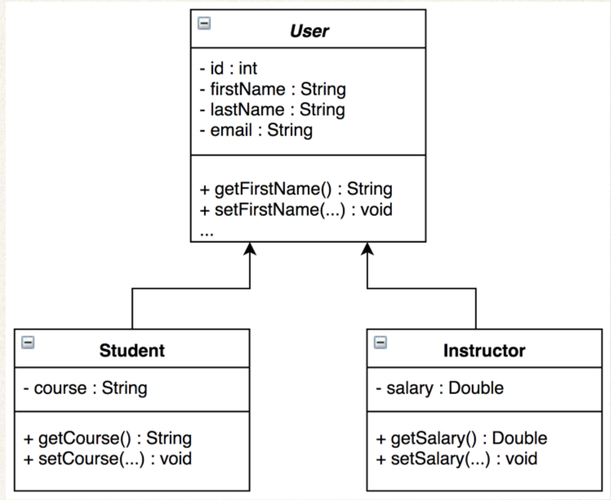
Run the App

Console

```
Hibernate: insert into Student (email, first_name, last_name, course, id) values (?, ?, ?, ?, ?)
Hibernate: insert into Instructor (email, first_name, last_name, salary, id) values (?, ?, ?, ?, ?)
```

Table: Instructor

id	email	first_name	last_name	salary
2	john@luv2code.com	John	Doe	20000



Includes fields defined in subclass and fields inherited from superclass

Run the App

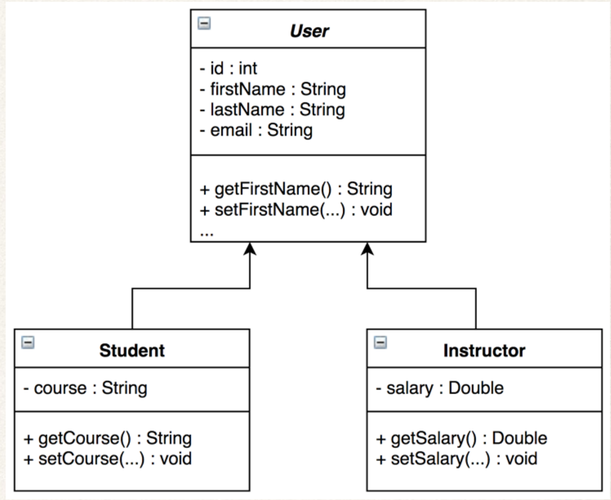
Console

```
Hibernate: insert into Student (email, first_name, last_name, course, id) values (?, ?, ?, ?, ?)
Hibernate: insert into Instructor (email, first_name, last_name, salary, id) values (?, ?, ?, ?, ?)
```

Table: Instructor

id	email	first_name	last_name	salary
2	john@luv2code.com	John	Doe	20000

User



Includes fields defined in subclass and fields inherited from superclass

Run the App

Console

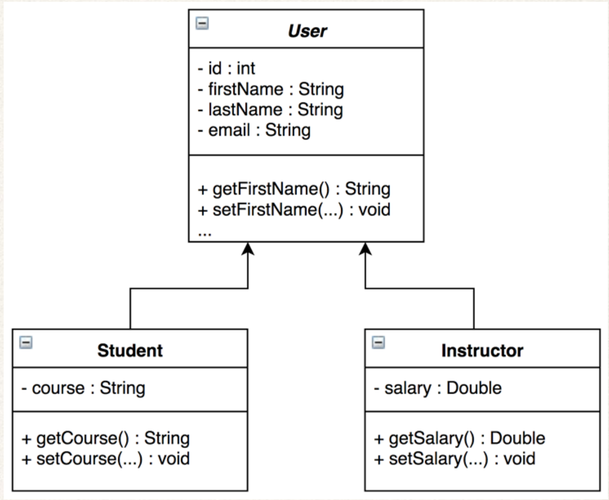
```
Hibernate: insert into Student (email, first_name, last_name, course, id) values (?, ?, ?, ?, ?)
Hibernate: insert into Instructor (email, first_name, last_name, salary, id) values (?, ?, ?, ?, ?)
```

Table: Instructor

id	email	first_name	last_name	salary
2	john@luv2code.com	John	Doe	20000

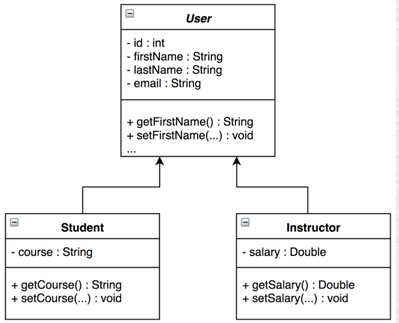
User

Instructor



Includes fields defined in subclass and fields inherited from superclass

ID from the Sequence Table



ID from the Sequence Table

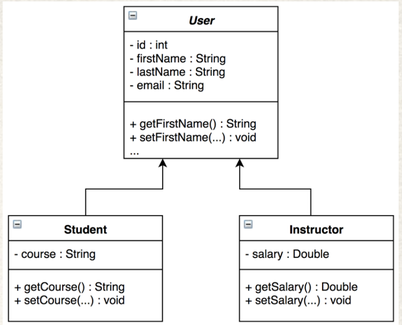


Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate

ID from the Sequence Table

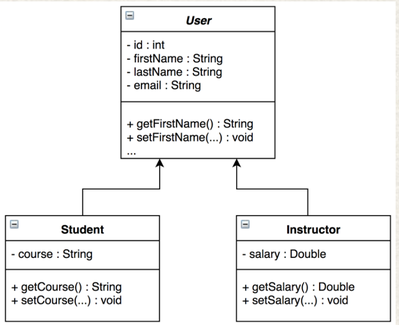


Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate

Table: Instructor

id	email	first_name	last_name	salary
2	john@luv2code.com	John	Doe	20000

ID from the Sequence Table

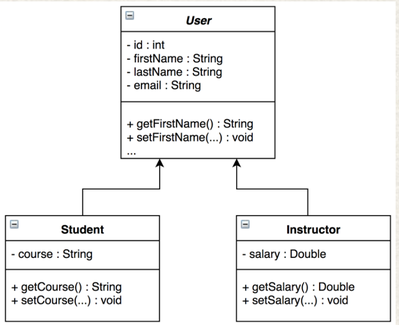


Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate

Table: Instructor

id	email	first_name	last_name	salary
2	john@luv2code.com	John	Doe	20000

```
// save the objects
System.out.println("Saving the student and instructor...");
session.save(tempStudent);
session.save(tempInstructor);
```

ID from the Sequence Table

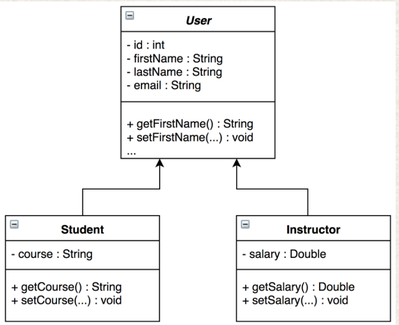


Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate

Table: Instructor

id	email	first_name	last_name	salary
2	john@luv2code.com	John	Doe	20000

```
// save the objects
System.out.println("Saving the student and instructor...");
session.save(tempStudent);
session.save(tempInstructor);
```

ID from the Sequence Table

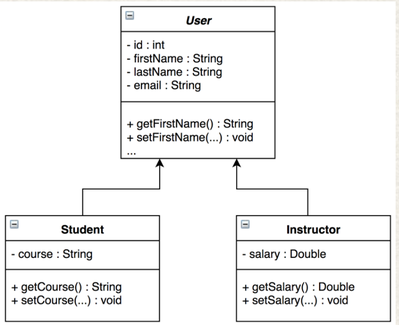


Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate

Table: Instructor

id	email	first_name	last_name	salary
2	john@luv2code.com	John	Doe	20000

```
// save the objects
System.out.println("Saving the student and instructor...");
session.save(tempStudent);
session.save(tempInstructor);
```

ID from the Sequence Table

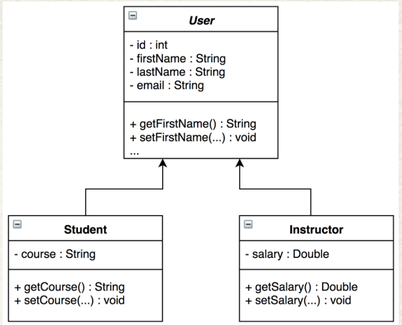


Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate

Table: Instructor

id	email	first_name	last_name	salary
2	john@luv2code.com	John	Doe	20000

```
// save the objects
System.out.println("Saving the student and instructor...");
session.save(tempStudent);
session.save(tempInstructor);
```

```
@Id
@GeneratedValue(strategy=GenerationType.TABLE)
@Column(name="id")
private int id;
```

ID from the Sequence Table

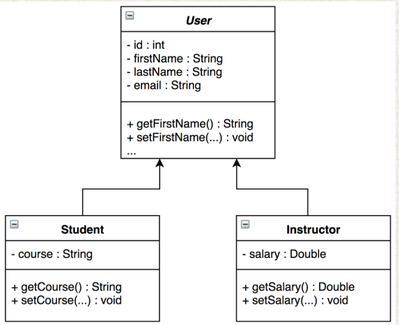


Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate

Table: Instructor

id	email	first_name	last_name	salary
2	john@luv2code.com	John	Doe	20000

```
// save the objects
System.out.println("Saving the student and instructor...");
session.save(tempStudent);
session.save(tempInstructor);
```

```
@Id
@GeneratedValue(strategy=GenerationType.TABLE)
@Column(name="id")
private int id;
```

ID from the Sequence Table

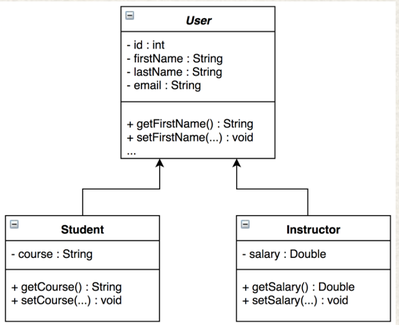


Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate

Table: Instructor

id	email	first_name	last_name	salary
2	john@luv2code.com	John	Doe	20000

```
2 •  SELECT * FROM hb_student_tracker.hibernate_sequences;
```

100% 1:2

Result Grid | Filter Rows: Search | Edit: | Export/Import:

sequence_name	next_val
default	2

```
// save the objects
System.out.println("Saving the student and instructor...");
session.save(tempStudent);
session.save(tempInstructor);
```

```
@Id
@GeneratedValue(strategy=GenerationType.TABLE)
@Column(name="id")
private int id;
```

ID from the Sequence Table

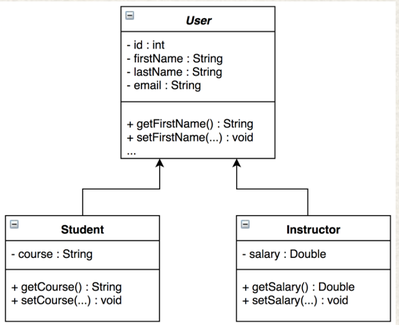


Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate

Table: Instructor

id	email	first_name	last_name	salary
2	john@luv2code.com	John	Doe	20000

```
// save the objects
System.out.println("Saving the student and instructor...");
session.save(tempStudent);
session.save(tempInstructor);
```

```
@Id
@GeneratedValue(strategy=GenerationType.TABLE)
@Column(name="id")
private int id;
```

```
2 • SELECT * FROM hb_student_tracker.hibernate_sequences;
```

100% 1:2

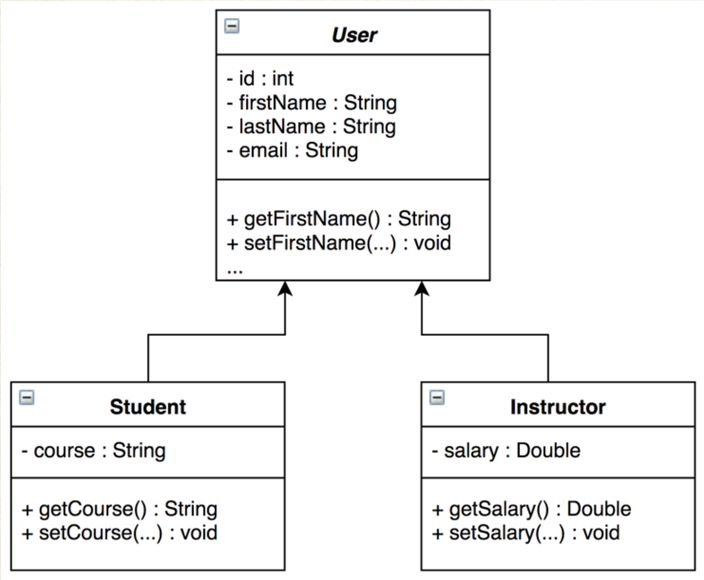
Result Grid | Filter Rows: Search | Edit: | Export/Import:

sequence_name	next_val
default	2

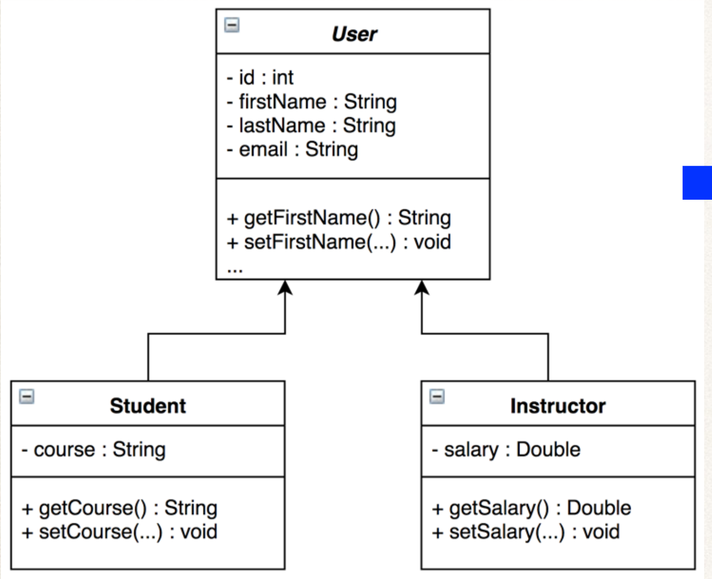
**Behind the scenes,
Hibernate will get the next value
from the sequence table**

**and then increment it by 1
(thread-safe)**

Inheritance Strategy - Table per Class

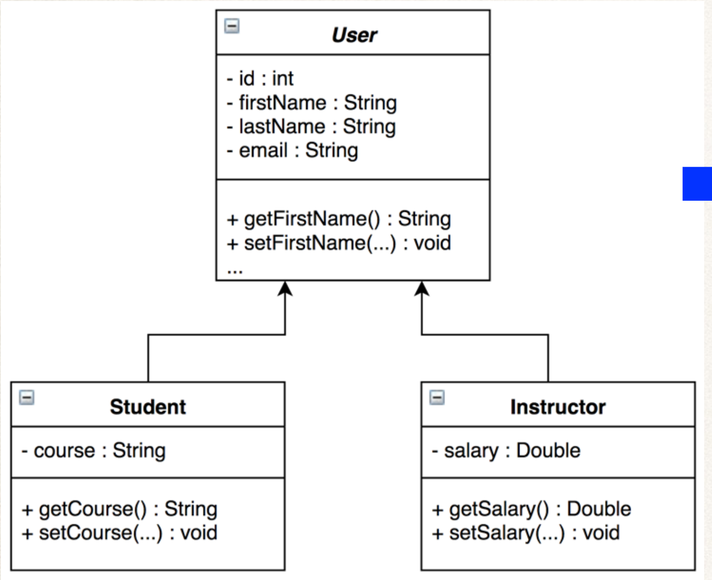


Inheritance Strategy - Table per Class



Inheritance Strategy - Table per Class

Table per class for the concrete classes



Inheritance Strategy - Table per Class

Table per class for the concrete classes

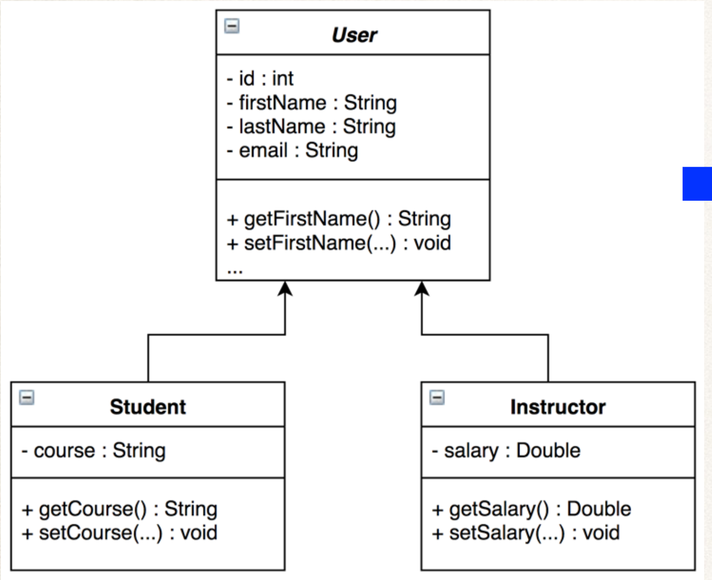


Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate

User

Student

Inheritance Strategy - Table per Class

Table per class for the concrete classes

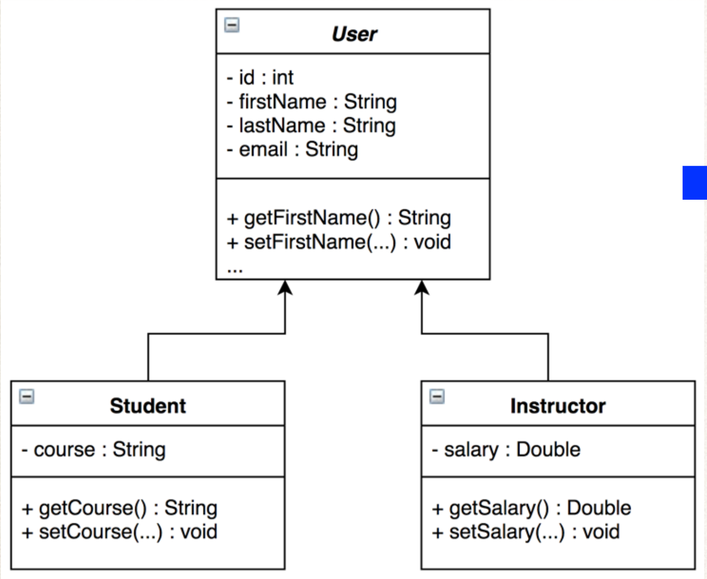


Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate

User

Student

Table: Instructor

id	email	first_name	last_name	salary
2	john@luv2code.com	John	Doe	20000

User

Instructor

Inheritance Strategy - Table per Class

Table per class for the concrete classes

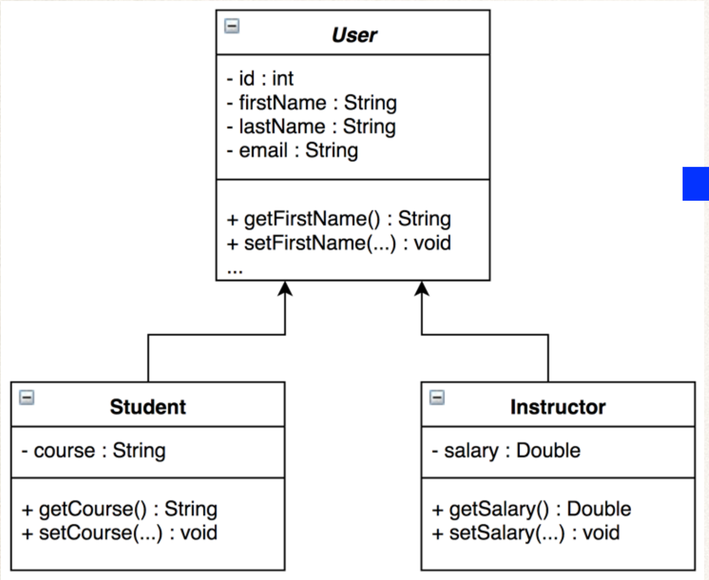


Table: Student

id	email	first_name	last_name	course
1	mary@luv2code.com	Mary	Public	Hibernate

User

Student

Table: Instructor

id	email	first_name	last_name	salary
2	john@luv2code.com	John	Doe	20000

User

Instructor

Note: Only the related fields
No extra columns

Table per Class

Table per Class

- Advantages

Table per Class

- Advantages
  - Simple and straight-forward implementation

Table per Class

- Advantages
  - Simple and straight-forward implementation
  - Queries on concrete sub-classes perform generally well

Table per Class

- Advantages
  - Simple and straight-forward implementation
  - Queries on concrete sub-classes perform generally well

- Disadvantages

Table per Class

- Advantages
  - Simple and straight-forward implementation
  - Queries on concrete sub-classes perform generally well

- Disadvantages
  - Slow performance for queries on superclasses (results in many joins)

Table per Class

- Advantages
  - Simple and straight-forward implementation
  - Queries on concrete sub-classes perform generally well

- Disadvantages
  - Slow performance for queries on superclasses (results in many joins)
  - Query performance slows down for very deep inheritance trees

Table per Class

- Advantages
  - Simple and straight-forward implementation
  - Queries on concrete sub-classes perform generally well

- Disadvantages
  - Slow performance for queries on superclasses (results in many joins)
  - Query performance slows down for very deep inheritance trees
  - ID generation with sequence tables in a high-volume multi-threaded environment is slow