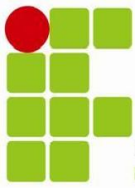


INSTITUTO FEDERAL
PARANÁ

MINISTÉRIO DA
EDUCAÇÃO

Como iniciar um projeto com o Hibernate

Conteúdo

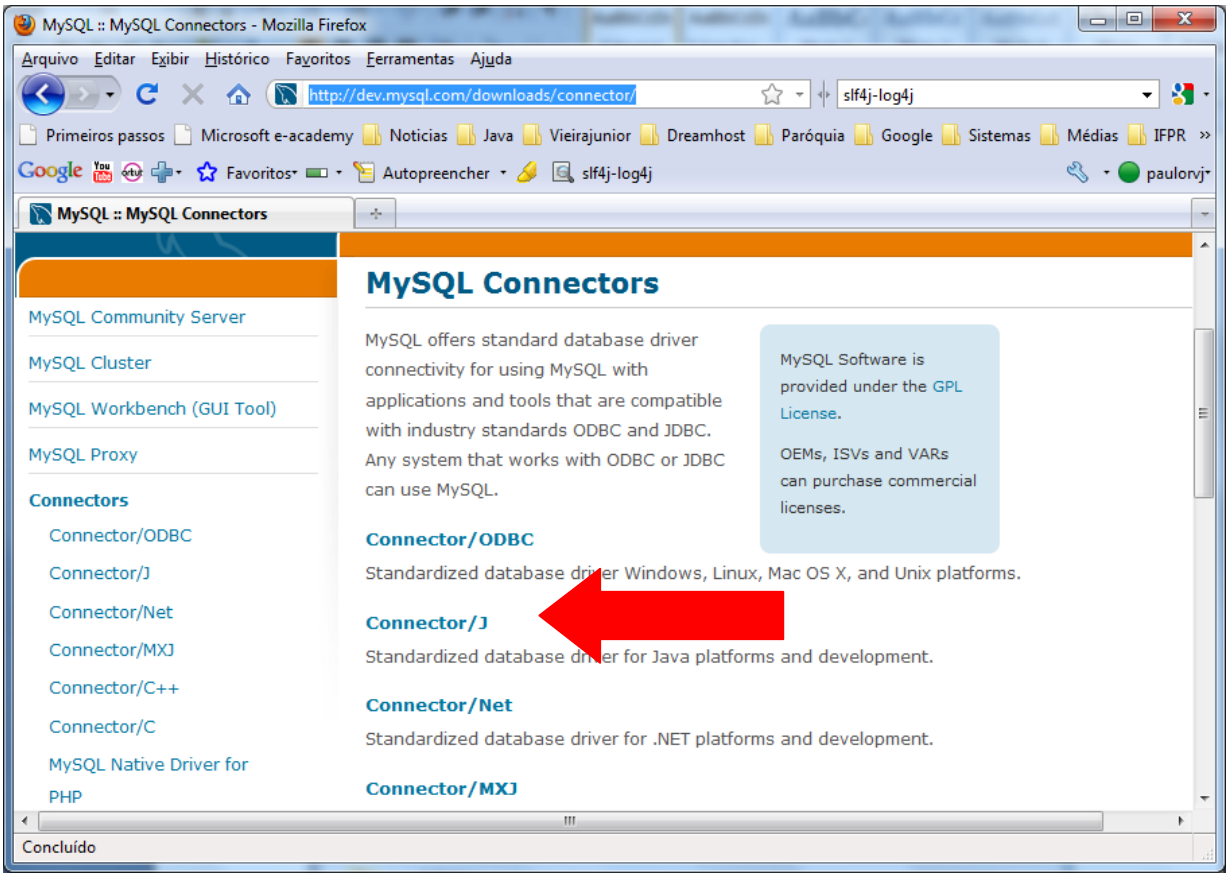
1 – Preparação e Downloads necessários

Antes de iniciar faça o download do Hibernate faça o download dos seguintes projeto:

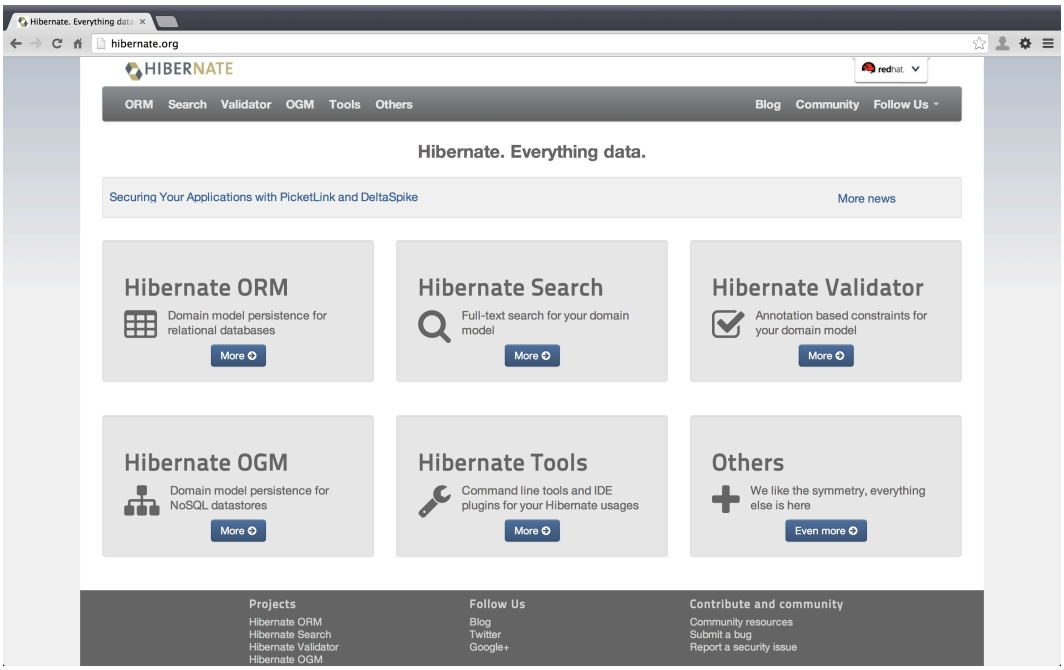
Entre no seguinte endereço e faça o download do Log4J versão 2.x:
http://logging.apache.org/log4j/2.x/

Vamos precisar também de um banco de dados, pode ser qualquer um, mas neste exemplo vou utilizar o Mysql que pode ser encontrado em www.mysql.com. Após efetuar o download do banco e instalar, precisamos também do Driver JDBC do banco. O driver JDBC do Mysql pode ser encontrado na sessão downloads em "Connectors".

http://dev.mysql.com/downloads/connector/



Agora faça o download do Hibernate ORM em http://hibernate.org/ :

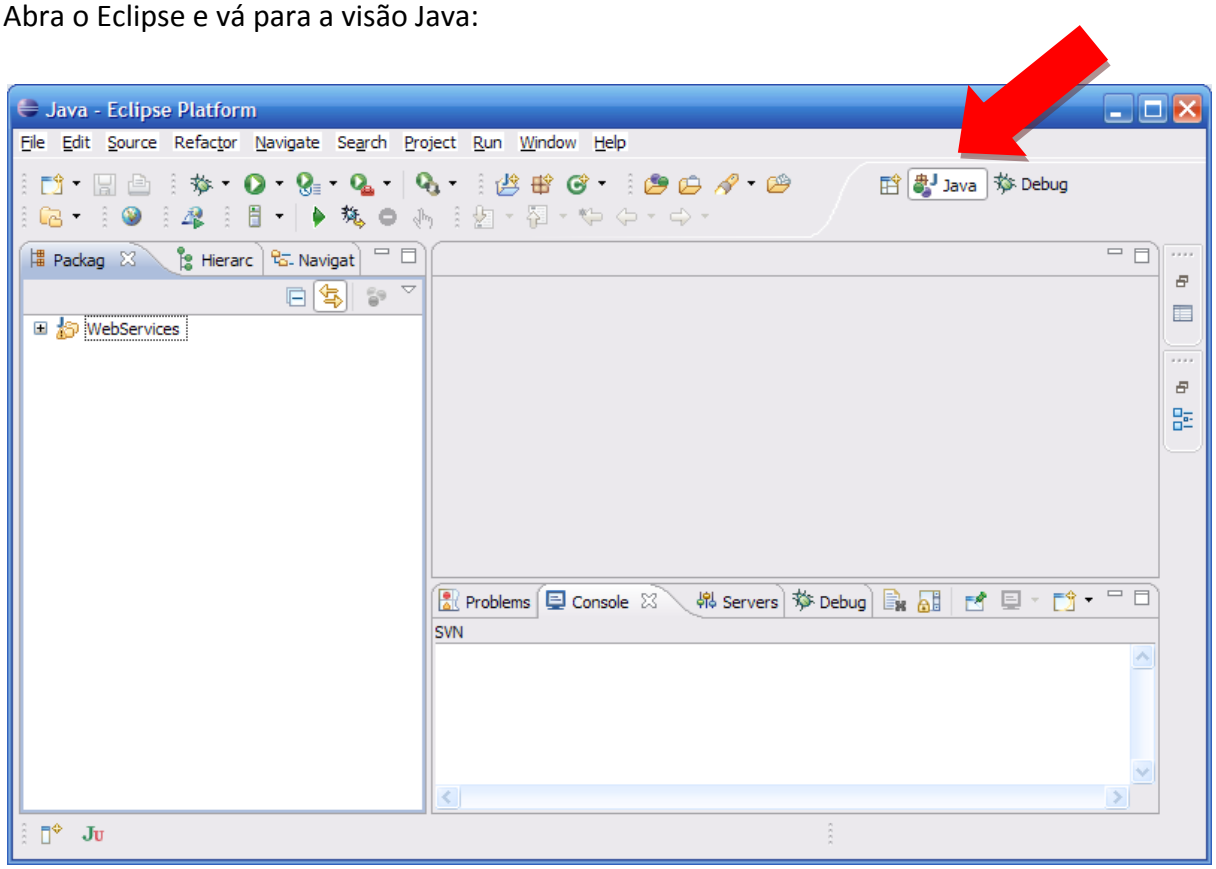


Descompacte os arquivos:

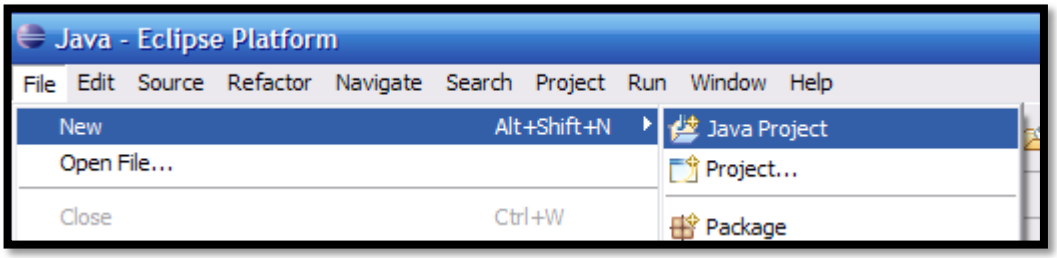
- Log4j
- Mysql Driver
- Hibernate

2 – Iniciando um projeto no Eclipse

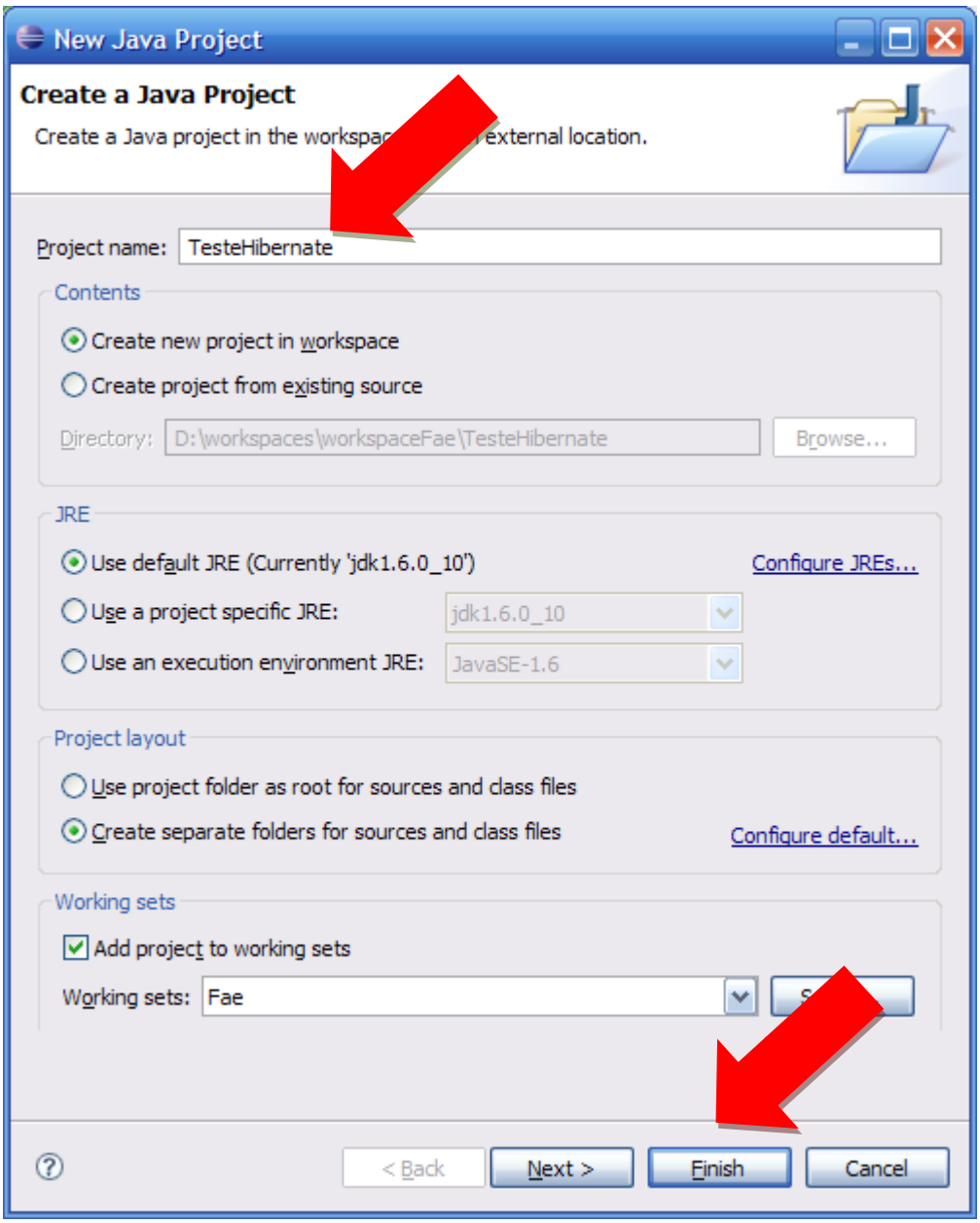
Neste exemplo vamos utilizar o Eclipse que pode ser encontrado em www.eclipse.org.
Abra o Eclipse e vá para a visão Java:



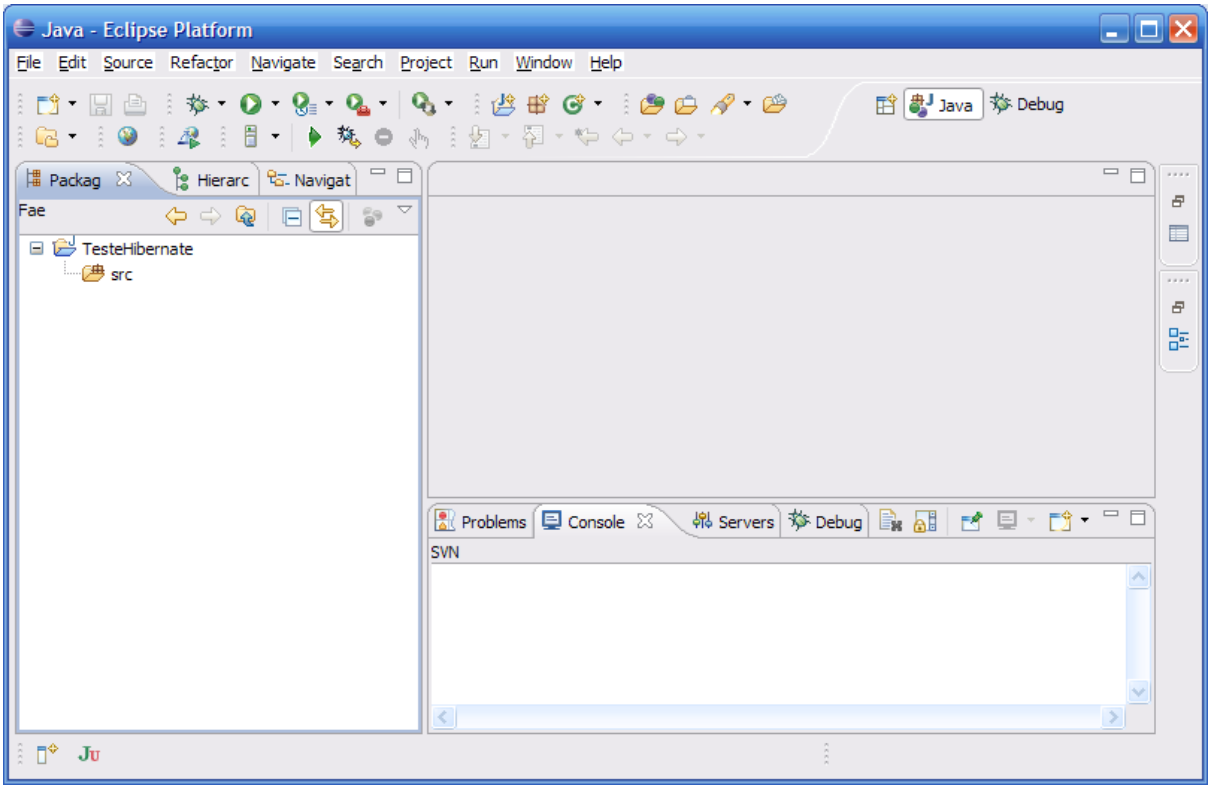
Vá no menu “File – New – Java Project”:



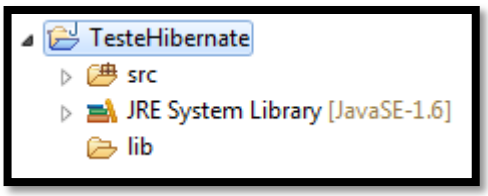
Vamos chamar o projeto de “TesteHibernate” e depois clique no botão “Finish”:



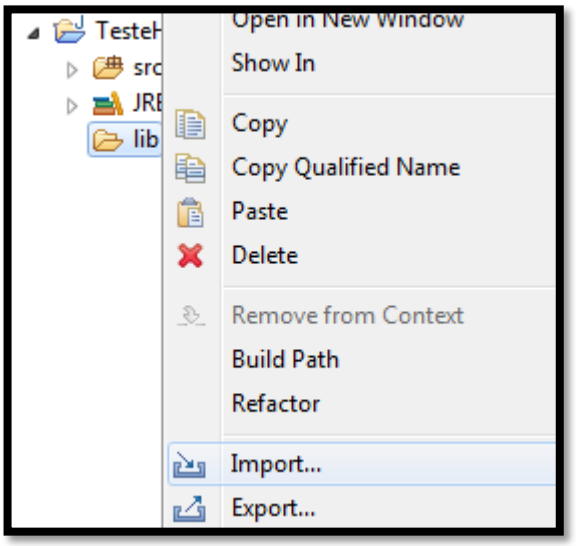
Seu Eclipse deve se parece com este:



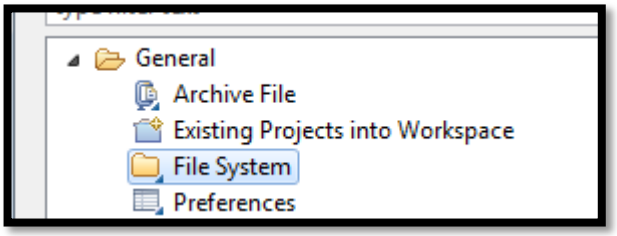
Dentro do diretório “src” é que irão ficar os pacotes, códigos Java e alguns arquivos de configuração do Hibernate.
O próximo passo é adicionar as libs necessárias no “Build Path” do projeto. Para isso vamos criar uma pasta chamada “lib” dentro do projeto e importar todos os arquivos JARs necessários:



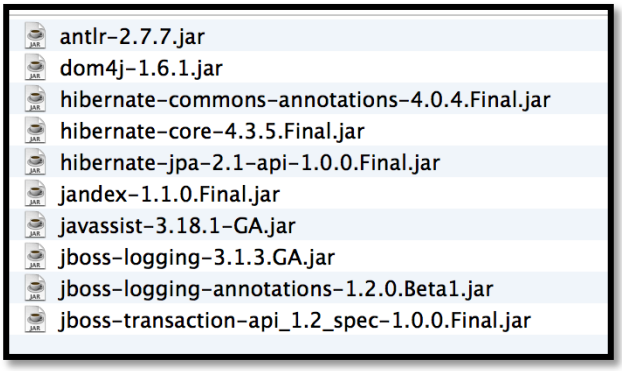
Clique com o botão direito na pasta “lib” e vá na opção import:



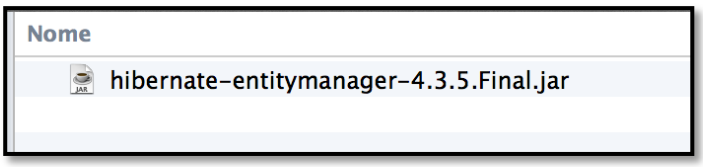
Selecione a opção “File System”:



Navegue até o diretório do hibernate e adicione as libs do diretório "***/hibernate-distribution\lib\required***" e selecione todos os arquivos desse diretório:

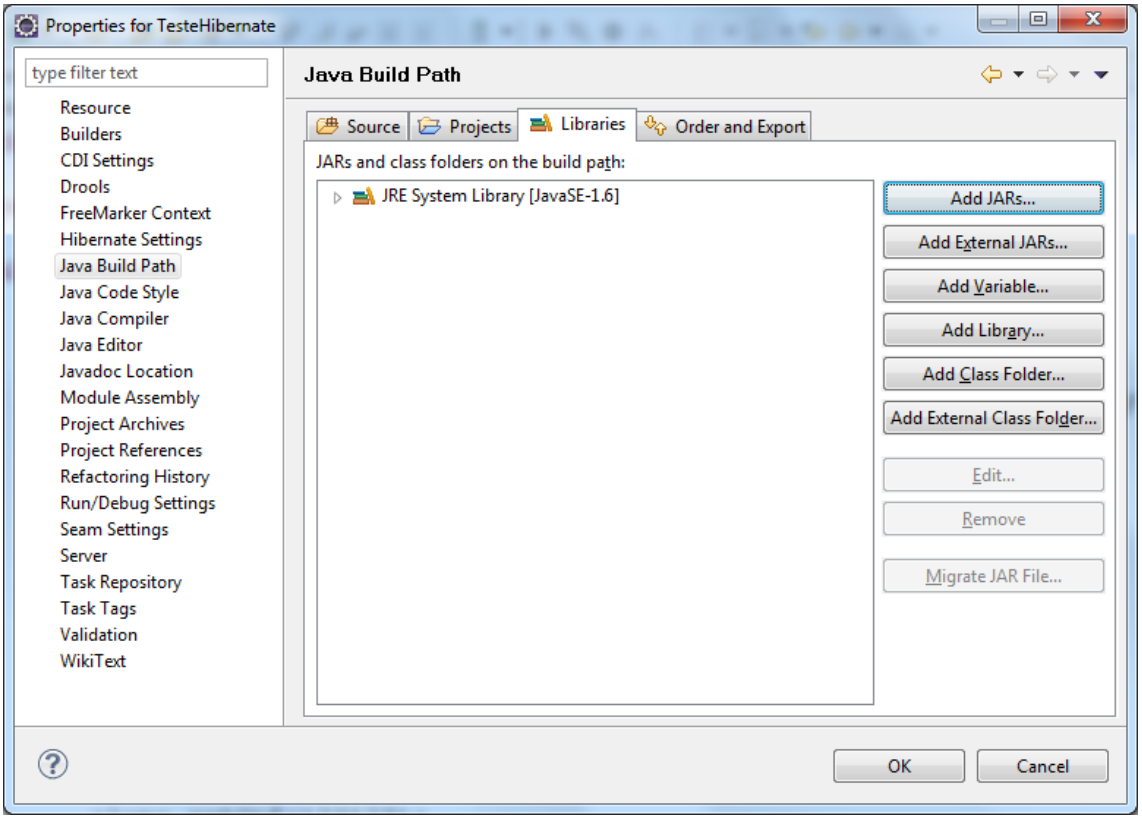


Adicione também o jar que está na pasta "jpa":

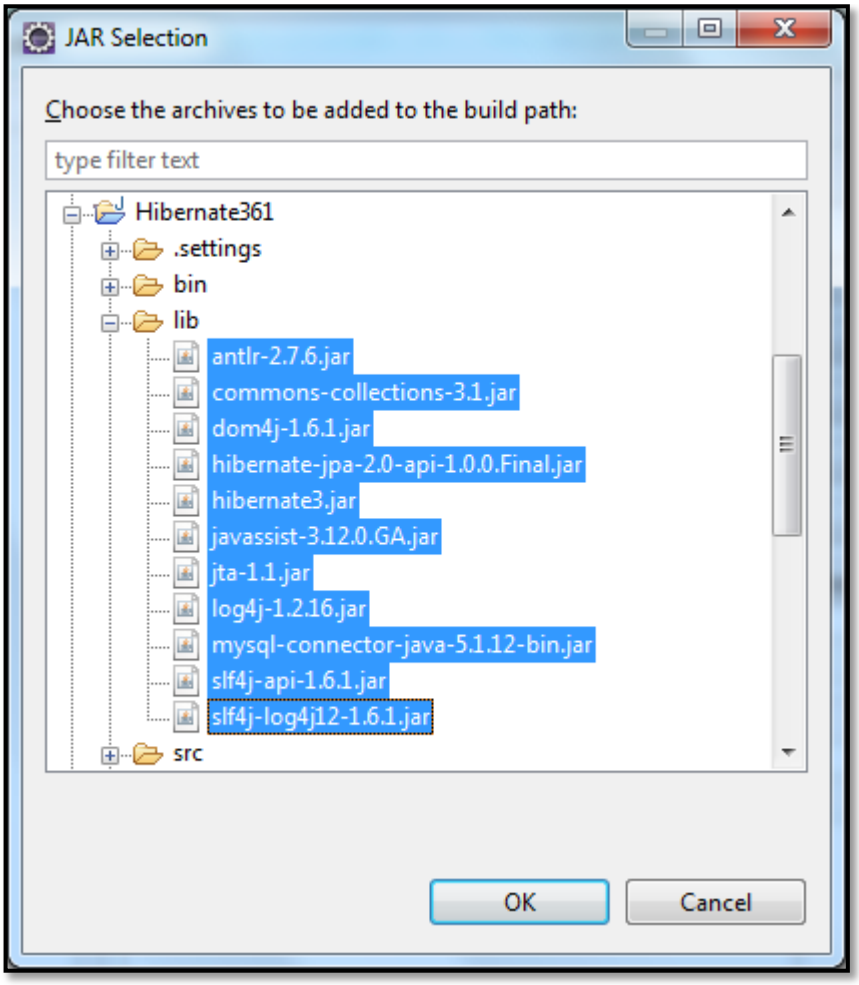


Adicione também o Log4J e o Driver Mysql.

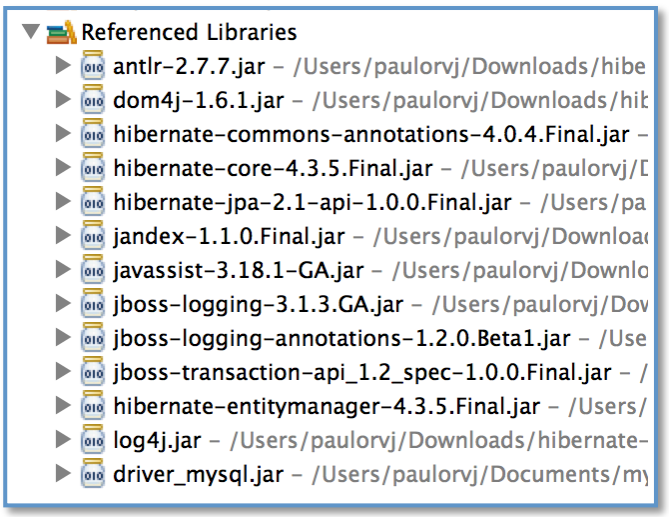
Agora precisamos configurar o projeto para utilizar todas as “Libs” importadas anteriormente. Clique com o botão direito do mouse no projeto e vá na opção “Properties”, procure na lista o item “Java Build Path” e vá na guia “libraries”:



Clique no botão “Add JARs...” navegue o diretório lib do projeto e selecione todos os arquivos e clique em “OK”:



Pronto. Seu projeto deve ter todos os JARs da lista abaixo, clique em OK:



3 - Anotações

As a*nnotations* no Java provêem dados sobre um programa em tempo de compilação, execução ou carregamento (class loading) e não é parte do mesmo, além, de não interferir na operação do código anotado.

As *annotations* são um poderoso recurso da linguagem que visa facilitar o desenvolvimento de sistemas e suas principais utilizações é na geração de código ou configuração de aplicações.

As a*nnotations* podem ser aplicadas na declaração de classes, campos, métodos e elementos de um programa.

As *annotations* provêem informações de várias maneira, dentre elas:

- **Informações para o compilador** — são usadas pelo compilador para detectar erros ou suprimi-los.
- **Tempo de compilação ou processamento de um deploy** — Neste caso as *annotations* substituem arquivos de configuração em XML.
- **Processamento em tempo de execução** — Algumas anotações estão disponiveis para serem examinadas em tempo de execução.

As anotações sempre começam com o símbolo @ “arroba” seguido de um nome:

Exemplos:

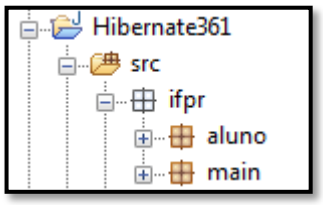
```
@Entity
class Person() {
}
```

OU

```
@SuppressWarnings("unchecked")
public  void setName(String name) {
        this.name = name;
}
```

4 – Criando e anotando classes

Vamos criar uma classe chamada Aluno, colocar alguns atributos e fazer o mapeamento Objeto/Relacional com anotações.
Primeiro crie um pacote chamado “ifpr” e dentro desse pacote crie outro pacote chamado “main” e outro chamado “aluno”.



Dentro do pacote “aluno” crie a classe Aluno com os atributos abaixo, não se esqueça de criar os métodos “gets” e “sets” para cada atributo:

```
1  package ifpr.aluno;
2
3  public class Aluno
4  {
5      private long id;
6      private String nome;
7      private String login;
8      private String senha;
9      private String dataNascimento;
```

Vamos anotar a nossa classe “Aluno”.
1 – A primeira anotação que vamos colocar é a anotação “`@Entity`” do pacote “`javax.persistence`” logo acima do nome da classe. Essa anotação diz para o hibernate que nossa classe é uma entidade e pode ser persistida no banco de dados.
2 – O hibernate pode criar a tabela automaticamente para nós, caso não especifiquemos um nome ele criará a tabela com o mesmo nome da classe. Vamos mudar o nome da tabela com a anotação “`@Table`”.
3 – Precisamos informar qual atributo será a nossa chave primária na tabela, neste caso o nosso atributo “id”, identificamos ele como chave primária colocando a anotação “`@Id`” logo acima do atributo.
4 – Precisamos informar também qual é a estratégia de geração da chave primária, neste caso como estamos utilizando o Mysql, vamos usar o autoincremento.
Adicionamos logo abaixo da anotação “`@Id`” a anotação “`@GeneratedValue(strategy=GenerationType.IDENTITY)`”.

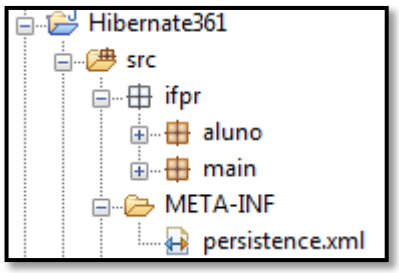
5 – Caso não especifiquemos os nomes das colunas o hibernate cria as colunas com o nome dos atributos. Vamos alterar o nome da coluna do atributo "dataDeNascimento", para isto, basta colocar a anotação "`@Column(name="data_nascimento")`".
6 – Vamos colocar uma restrição no atributo matricula, ele deve ter tamanho máximo de 20 caracteres e não pode ser nulo. Utilizamos a mesma anotação do item 5, "`@Column(nullable=false, length=20)`".

Sua classe deve estar parecida com esta:

```
package ifpr.aluno;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="tbAluno")
public class Aluno
{
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private long id;

	@Column(nullable=false)
	private String nome;

	private String login;
	private String senha;

	@Column(name="data_nascimento")
	private String dataNascimento;

	// gets e sets

}
```

5 – Configurando o Hibernate

Para que o hibernate funcione adequadamente, precisamos configurá-lo. **Dentro do diretório "src"** vamos criar uma pasta com o nome "META-INF" desse jeito mesmo, com as letras em maiúsculas. Dentro dessa pasta vamos criar um arquivo XML com o nome "persistence.xml":



Seu conteúdo deve ser o seguinte:

```
1<persistence xmlns="http://java.sun.com/xml/ns/persistence"
2    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
4            http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
5    version="1.0">
6    <persistence-unit name="testePU" transaction-type="RESOURCE_LOCAL">
7        <provider>org.hibernate.ejb.HibernatePersistence</provider>
8        <properties>
9            <property name="hibernate.show_sql" value="true"/>
10           <property name="hibernate.hbm2ddl.auto" value="create"/>
11           <property name="hibernate.query.substitutions" value="true=1, false=0"/>
12
13           <property name="hibernate.connection.driver_class"
14               value="com.mysql.jdbc.Driver"/>
15           <property name="hibernate.connection.url"
16               value="jdbc:mysql://localhost:3306/testehibernate"/>
17           <property name="hibernate.connection.username"
18               value="teste"/>
19           <property name="hibernate.connection.password"
20               value="teste01"/>
21           <property name="hibernate.dialect"
22               value="org.hibernate.dialect.MySQLDialect"/>
23       </properties>
24   </persistence-unit>
25</persistence>
```

Na linha 6 estamos definindo o nome de nossa unidade de persistência.
Na linha 9 estamos dizendo para o Hibernate mostrar os Sqls que ele gera.

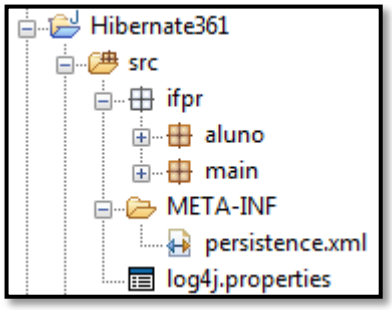
Na linha 10 estamos definindo o que o Hibernate vai fazer quando a aplicação iniciar. Podemos definir este campo com os valores:

- “create” – o Hibernate irá criar as tabelas na inicialização.
- “update” – o Hibernate irá atualizar o banco. Se você adicionou alguma classe ou algum atributo em alguma classe o Hibernate irá criar ou incluir as novas informações no banco de dados, porém, se você removeu alguma classe ou algum atributo, o Hibernate não irá remover as tabelas ou campos do banco.
- “validate” – o Hibernate apenas valida o schemma do banco em relação às entidades.
- “create-drop” – o Hibernate cria as tabelas e deleta após a execução.
- None – não faz nada.

Nas linhas 13 e 14 estamos definindo a classe do driver de conexão com o banco.
Nas linhas 15 e 16 a url do banco com o nome do database que será utilizado.
Nas linhas 17 e 18 o nome do usuário para conectar no banco.
Nas linhas 19 e 20 a senha do usuário para conectar no banco.
E por fim nas linhas 21 e 22 o dialeto que Hibernate irá utilizar para as consultas e seu valor muda conforme o banco de dados que está sendo utilizado.

6 – Configurando o Log4j

Para configurar o Log4j basta criar um arquivo com o nome “log4j.properties” dentro da pasta “src”, o arquivo deve ter o seguinte conteúdo:




```
#log4j.rootLogger=ERROR,dest1
log4j.rootLogger=DEBUG,dest1

log4j.appender.dest1=org.apache.log4j.ConsoleAppender
log4j.appender.dest1.layout=org.apache.log4j.PatternLayout
log4j.appender.dest1.layout.ConversionPattern=%d  %-5p  %-5c{3}  %x  ->
%m%n

#log4j.appender.dest2=org.apache.log4j.RollingFileAppender
#log4j.appender.dest2.File=bridge.log

#log4j.appender.dest2.MaxFileSize=100KB
# Keep one backup file
#log4j.appender.dest2.MaxBackupIndex=3

#log4j.appender.dest2.layout=org.apache.log4j.PatternLayout
#log4j.appender.dest2.layout.ConversionPattern=%d    [%t]    %-5p    %-
5c{3}(%L) %x -> %m%n
```

7 – Efetuando algumas operações

Adicionar aluno

Vamos começar criando um novo aluno. Crie uma classe chamada “AdicionarAluno” dentro pacote “main” com o seguinte conteúdo:

```
1.    package ifpr.main;
2.
3.    import javax.persistence.EntityManager;
4.    import javax.persistence.EntityManagerFactory;
5.    import javax.persistence.Persistence;
6.
7.    public class AdicionarAluno {
8.        public static void main(String[] args) {
9.            EntityManagerFactory emf = Persistence.
10.                                createEntityManagerFacto-
ry("testePU");
11.           EntityManager em = emf.createEntityManager();
12.
13.           Aluno aluno = new Aluno();
14.           aluno.setNome("Paulo");
15.           aluno.setDataDeNascimento("09/05/1978");
16.           aluno.setMatricula("71623764128");
17.           aluno.setLogin("paulo");
18.
19.           em.getTransaction().begin();
20.           em.persist(aluno);
21.           em.getTransaction().commit();
22.
23.           em.close();
24.           emf.close();
25.       }
26.   }
```

Na linha 9 estamos criando um objeto EntityManagerFactory informando o nome de nossa unidade de persistência. A unidade de persistência é o nome que configuramos dentro do arquivo “persistence.xml”.
Na linha 11 estamos criando um objeto EntityManager, é este objeto que irá gerenciar a nossa entidade Aluno.
Nas linhas 13 a 17 estamos criando um objeto da classe aluno.
Na linha 19 estamos pedindo para o EntityManager criar uma transação.
Na linha 20 estamos pedindo para o EntityManager persistir o objeto aluno no banco de dados.
Na linha 21 estamos efetuando a transação.
Nas 23 e 24, estamos fechando o EntityManager e a conexão com o banco.

Atualizar um aluno

Crie uma classe chamada “AtualizarAluno” dentro do pacote “main” com o seguinte conteúdo:

```
1.    package ifpr.main;
2.
3.    import javax.persistence.EntityManager;
4.    import javax.persistence.EntityManagerFactory;
5.    import javax.persistence.Persistence;
6.
7.    public class AtualizarAluno
8.    {
9.        public static void main(String[] args)
10.       {
11.           EntityManagerFactory emf = Persistence.
12.                                   createEntityManagerFacto-
ry("testePU");
13.           EntityManager em = emf.createEntityManager();
14.
15.           Aluno aluno = em.find(Aluno.class, 2L);
16.           aluno.setLogin("paulo2");
17.
18.           em.getTransaction().begin();
19.           em.merge(aluno);
20.           em.getTransaction().commit();
21.
22.           em.close();
23.           emf.close();
24.       }
25.   }
```

Na linha 15 estamos pedindo para o EntityManager buscar um objeto da classe “Aluno” com o id do tipo long 2. O Hibernate faz a pesquisa e retorna o objeto.
Na linha 16 estamos alterando o valor do atributo login.
Na linha 18 estamos pedindo para o EntityManager criar uma transação.
Na linha 19 estamos pedindo para o EntityManager atualizar o objeto aluno no banco de dados.
Na linha 20 estamos efetuando a transação.
Nas 22 e 23, estamos fechando o EntityManager e a conexão com o banco.

Remover um aluno

Crie uma classe chamada “RemoverAluno” dentro do pacote “main” com o seguinte conteúdo:

```
1. package ifpr.main;
2.
3. import javax.persistence.EntityManager;
4. import javax.persistence.EntityManagerFactory;
5. import javax.persistence.Persistence;
6.
7. public class RemoverAluno
8. {
9.    public static void main(String[] args)
10.   {
11.       EntityManagerFactory emf = Persistence.
12.                                  createEntityManagerFacto-
   ry("testePU");
13.       EntityManager em = emf.createEntityManager();
14.
15.       Aluno aluno = em.find(Aluno.class, 1L);
16.
17.       em.getTransaction().begin();
18.       em.remove(aluno);
19.       em.getTransaction().commit();

20.       em.close();
21.       emf.close();
22.    }
23.   }
```

Na linha 15 estamos pedindo para o EntityManager buscar um objeto da classe “Aluno” com o id do tipo long 1. O Hibernate faz a pesquisa e retorna o objeto.
Na linha 17 estamos pedindo para o EntityManager criar uma transação.
Na linha 18 estamos pedindo para o EntityManager remover o objeto aluno no banco de dados.
Na linha 19 estamos efetuando a transação.
Nas 20 e 21, estamos fechando o EntityManager e a conexão com o banco.