JAVA FRAMEWORK

Java dünyasındaki frameworklere geçmeden önce “framework” tanımını yapmamız gerekiyor. Frameworkler uygulama veya yazılım geliştirmek için geliştirilen ve kullanıma sunulan araçlardır. Sundukları farklı avantajlar ile birlikte geliştiricilerin işlerini kolaylaştırırlar.

Java için birçok geliştirilmiş framework vardır. Ben bu yazı ile 2 tanesini açıklamak istiyorum. Bunlar:

1-Spring

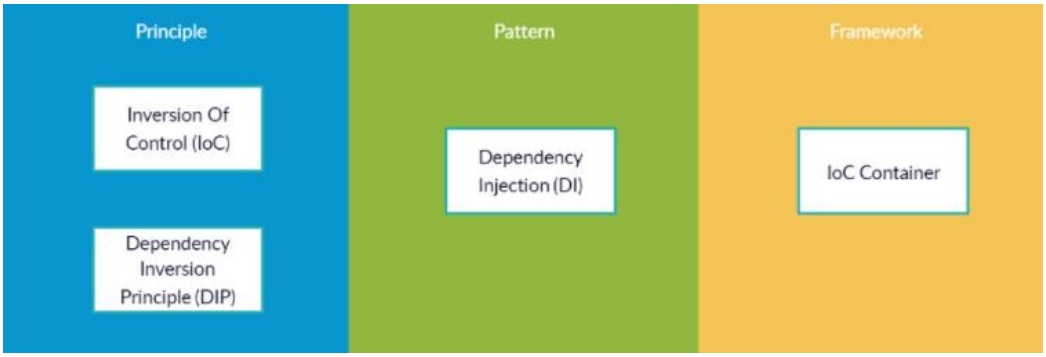
2-Hibernate

SPRING FRAMEWORK

Spring, kurumsal olarak uygulamaların geliştirilmesinde ve karmaşıklığı azaltmak için kullanılan çok yönlü Java frameworklerinden biridir. Açık kaynak olmasının yanında lightweight (hafif) olması ve çok sayıda Java uygulaması bu framework yapısının temel özelliklerini kullanır. Web uygulama geliştirme özelliklerini desteklerken, JEE (Enterpries Java Applications) kurumsal java uygulamalarının oluşturulmasında yardımcı olur.

Ayrıca çeşitli frameworkler ile sağlıklı bir şekilde çalıştığı için bir framework çatısı olarak düşünülebilir.

Üretkenliği arttırmak ve hataları en aza indirmek için Java Veritabanı Bağlantısı (Java Database Connectivity) desteğini sağlar. Bunların yanı sıra yazılan kodların uyumluluğu ve test edilebilirliği konusunda büyük bir avantaj sağlar. Ayrıca Spring, kendi container servisleri ile IoC yani Inversion of Control denilen tasarım desenini kullanır. “Dependency Injection” (bağımlılık enjeksiyonu) ile uygulama akışında esneklik sağlar. Aşağıdaki görsel ile çok karıştırılan kavramlar belirtilmiştir.



IoC ve Dependency Injection ile Bağımlılıkları Ortadan Kaldırma

```
class Company {
  Address address;

  Company() {
    address = new Address();
  }
}
```

Yukarıda bulunan kodda Company sınıfı ile “Address” sıkı bağımlı şekildedir. Fakat kodumuzu aşağıdaki şekilde düzeltirsek bağımlılığı azaltırız.

```
class Company{
  Address address;

  Company(Address address){
      this.address=address;
      }
}
```

Spring, bunu yapısında bulunan IoC containerlar sorumludur. Bağımlılıkları azaltarak kodumuzun test edilebilirliğini kolaylaştırırız.

SPRING FRAMEWORK KOD ÖRNEK

Spring framework kullanarak uygulama yapmak için ilk olarak Java sınıfımızı oluştururuz.

```
public class Game {

   private String name;


public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public void helloSpring(){
    System.out.println("Hello: "+name);
}
}
```

Burada Game ismi ile bir sınıf oluşturduk. Getter ve Setter kullandık ve helloSpring metodu içerisinde oyunun ismini yazdırdık.

Sonrasında applicationContext.xml dosyası içerisinde “bean” oluşturarak bu sınıfımızı Spring tarafına tanıtacağız.

```
<bean id="studentbean" class="com.javatpoint.Student">
<property name="name" value="league of legends"></property>
</bean>
```

Sonrasında test için bir Test sınıfı oluşturup Spring framework çalışmasına bakabiliriz.

```
public class Test {
public static void main(String[] args) {

    Resource resource=new ClassPathResource("applicationContext.xml");
    BeanFactory factory=new XmlBeanFactory(resource);

    Game game=(Game)factory.getBean("gamebean");
    game.helloSpring();
}
}
```

Resource nesnesi ile applicationContext.xml dosya bilgileri için oluşturulmuştur.

Bean tarafından sorumlu olan BeanFactory’dir. Burada birçok metod bulunmaktadır. Game nesnesinin dönmesinden (yani get) sorumlu olarak getBean kullanılmıştır.

```
Hello: league of legends
```

Test ile bu çıktıyı elde edebiliriz.

HIBERNATE FRAMEWORK

Hibernate frameworkü, Java için nesne – ilişkisel (object-relational) eşleme aracı olarak tanımlanır. Nesne – ilişkisel uyumsuzluk sorunlarına çözüm arar. Veri kalıcılığı için Java Persistence API özellikleri uygular.

Java tarafında oluşturulan sınıflar ile veritabanı tablolarına mapping işlemi yaparken aynı zamanda Java veri türleri ile SQL veri türleri arasında mapping yapabilir. Veri tabanını otomatik olarak oluşturarak manuel olarak veri tabanı oluşturmamıza gerek yoktur.

Hız açısından büyük bir avantaj sağlar. Bunun nedeni önbellek kullanımını dahili olarak kullanır. Veri tabanlarından bağımsız olarak sorgu yapabilmesi geliştiricilerin işini kolaylaştıran başka bir Hibernate framework avantajıdır.

HIBERNATE KOD ORNEK

Basit bir kalıcı sınıf oluşturmamız gerekiyor. Kalıcı (persistence) sınıfın bazı kuralları vardır. Constructor bulunmaz. Private olarak özellik tutar. Getter ve Setter methodları bulunur.

```
public class Student {

    private int id;
    private String firstName,lastName;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
}
```

Yukarıda “Student” isimli sınıfımızı oluşturduk. Sonrasında Hibernate için xml dosyası oluşturulur.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
 "-//Hibernate/Hibernate Mapping DTD 5.3//EN"
 "http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

 <hibernate-mapping>
  <class name="com.(yourpackage).Student" table="std1000">
    <id name="id">
     <generator class="assigned"></generator>
    </id>

    <property name="firstName"></property>
    <property name="lastName"></property>

  </class>

</hibernate-mapping>
```

Burada olduğu gibi isim oluşturulan sınıf ile aynı olmalıdır. Bu xml dosyasında bulunan ifadeleri açıklayacak olursak;

hibernate-mapping : içerisinde eşleme dosyasındaki tüm eşleme ögelerini barındırır.

class : Oluşturduğumuz kalıcı sınıfı temsil eder.

id : Sınıf içerisinde bulunan birincil özelliği temsil eder.

property : Sınıfın diğer ögelerini belirtir.

Daha sonra Hibernate için bir yapılandırma dosyası oluştururuz. Bu dosyası aşağıda görebiliriz.

```xml
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 5.3//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">

<hibernate-configuration>

    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
        <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
        <property name="connection.username">system</property>
        <property name="connection.password">jtp</property>
        <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <mapping resource="student.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```

Sonrasında nesnemizi tutacağımız ve depo olarak düşünebileceğimiz bir sınıf oluştururuz.

```java
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;


public class StoreMyClass {
public static void main(String[] args) {

    //Create typesafe ServiceRegistry object
    StandardServiceRegistry ssr = new StandardServiceRegistryBuilder().configure
    ("hibernate.cfg.xml").build();

   Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();

SessionFactory factory = meta.getSessionFactoryBuilder().build();
Session session = factory.openSession();
Transaction t = session.beginTransaction();

    Employee e1=new Student();
    e1.setId(1);
    e1.setFirstName("Batuhan");
    e1.setLastName("Batu");

    session.save(e1);
    t.commit();
    System.out.println("Başarı ile kaydedildi");
    factory.close();
    session.close();

}
}
```

```
Başarı ile kaydedildi.
```

Çıktısını elde ederiz.