1-

1. Spring Framework

- Spring Framework, bağımlılık yönetimi (Dependency Injection) ile nesneler arasındaki bağımlılıkları yönetir, veri erişimi için JDBC ve ORM (Object-Relational Mapping) desteği sağlar. Ayrıca, web uygulamaları geliştirmeyi kolaylaştırır ve Spring Security ile güvenliği sağlar. Bu özellikler, Spring'i esnek ve modüler bir uygulama geliştirme platformu haline getirir.

Örnek: Spring Boot ile Basit REST API

UserController sınıfı:

```
@RestController
@RequestMapping("/api")
public class UserController {

    @Autowired
    private UserService userService;

    @GetMapping("/users")
    public List<User> getAllUsers() {
        return userService.findAllUsers();
    }

    @PostMapping("/users")
    public User createUser(@RequestBody User user) {
        return userService.saveUser(user);
    }
}
```

2. Hibernate

- Hibernate, nesne-ilişkisel eşleme (ORM) sağlayarak Java nesnelerini veritabanı tablolarına dönüştürür ve veri erişimini nesnelerle yönetir. Bu, veritabanı işlemlerini daha doğal ve nesne odaklı bir şekilde gerçekleştirmeyi mümkün kılar.

Örnek: Hibernate ile Entity Sınıfı ve CRUD İşlemleri

User entity sınıfı:

```
@Entity

public class User {


   @Id

   @GeneratedValue(strategy = GenerationType.IDENTITY)

   private Long id;

   private String email;

   private String password;


}
```

3. Apache Struts

- Apache Struts, Model-View-Controller (MVC) yapısını kullanarak web uygulamaları geliştirmenizi sağlar. Bu yapı, uygulamanın iş mantığını, kullanıcı arayüzünü ve kontrol akışını ayrı katmanlara ayırarak, daha düzenli ve bakımı kolay kod yazılmasını destekler.

Örnek: Struts2 ile Basit Bir Form İşleme

struts.xml yapılandırması:

```
<struts>

   <package name="user" namespace="/" extends="struts-default">

      <action name="register" class="com.example.RegisterAction">

         <result name="success">/success.jsp</result>
```

```
        <result name="input">/register.jsp</result>
    </action>
  </package>
</struts>
```

RegisterAction sınıfı:

```
public class RegisterAction extends ActionSupport {

  private String username;
  private String password;

  public String execute() {
    return SUCCESS;
  }
}
```

4. Apache Camel

- Apache Camel, çeşitli uygulama ve sistemler arasında entegrasyon ve mesaj yönlendirme işlemlerini kolaylaştıran bir açık kaynaklı framework'tür. Camel, çeşitli protokoller ve veri formatları arasında veri transferini yönetir ve entegrasyon süreçlerini basit ve ölçeklenebilir bir şekilde yapılandırmayı sağlar.

Örnek: Camel ile Dosya Yönlendirme

```
public class FileRouteBuilder extends RouteBuilder {

  @Override
  public void configure() throws Exception {
    from("file:input")
```

```
        .to("file:output");
    }
}
```

MainApp sınıfı:

```
public class MainApp {

    public static void main(String[] args) throws Exception {
        CamelContext context = new DefaultCamelContext();
        context.addRoutes(new FileRouteBuilder());
        context.start();
        Thread.sleep(5000);
        context.stop();
    }
}
```

5. JUnit

- JUnit, Java uygulamalarında birim testleri yazma ve yönetme için kullanılan bir test çerçevesidir. JUnit, test senaryolarını tanımlamanızı ve otomatik olarak çalıştırmanızı sağlayarak, kodunuzun doğruluğunu ve güvenilirliğini artırır.

Örnek: JUnit ile Basit Bir Test

```
public class UserServiceTest {

    private UserService userService;
```

```
    @Before
    public void setUp() {
        userService = new UserService();
    }

    @Test
    public void testSaveUser() {
        User user = new User("test@example.com", "password");
        userService.saveUser(user);
        assertNotNull(user.getId());
    }
}
```

2- SOA (Service-Oriented Architecture)

Tanım: SOA, yazılım bileşenlerinin birbirleriyle servisler aracılığıyla iletişim kurduğu bir yazılım tasarım yaklaşımıdır.

Örnek: Bir e-ticaret uygulamasında, envanter yönetimi, sipariş yönetimi ve kullanıcı yönetimi gibi farklı servisler SOA kullanılarak bağımsız ve birlikte çalışabilir şekilde tasarlanır.

Web Service

Tanım: Web servisleri, uygulamaların web üzerinden iletişim kurmasını sağlayan hizmetlerdir. XML, SOAP, WSDL gibi standartları kullanarak veri alışverişi yapar.

Örnek: Bir hava durumu web servisi, belirli bir konumun hava durumu bilgisini XML formatında döner.

```
<Weather>
    <Location>New York</Location>
    <Temperature>30</Temperature>
    <Condition>Sunny</Condition>
</Weather>
```

RESTful Service

Tanım: RESTful servisler, HTTP protokolünü kullanarak veri alışverişi yapar ve REST mimari tarzını uygular. JSON veya XML formatında veri döner.

Örnek: Bir kitap yönetim sistemi API'si.

- **GET /books**: Tüm kitapları getirir.
- **POST /books**: Yeni bir kitap ekler.
- **GET /books/{id}**: Belirli bir kitabı getirir.
- **PUT /books/{id}**: Belirli bir kitabı günceller.
- **DELETE /books/{id}**: Belirli bir kitabı siler.

```
{
 "id": 1,
 "title": "Java Programming",
 "author": "John Doe"
}
```

HTTP Methods

Tanım: HTTP metotları, web servisleriyle etkileşim kurarken kullanılan temel işlemleri tanımlar.

Örnekler:

- **GET:** Sunucudan veri almak için kullanılır.

```
GET /api/users HTTP/1.1
Host: example.com
```

- **POST:** Sunucuya veri göndermek için kullanılır.

```
POST /api/users HTTP/1.1
Host: example.com
Content-Type: application/json

{
 "name": "Alice",
 "email": "alice@example.com"
}
```

- **PUT:** Sunucudaki mevcut veriyi güncellemek için kullanılır.

```
PUT /api/users/1 HTTP/1.1
Host: example.com
Content-Type: application/json
```

```
{
  "name": "Alice Smith",
  "email": "alice.smith@example.com"
}
```

- **DELETE:** Sunucudaki veriyi silmek için kullanılır.

```
DELETE /api/users/1 HTTP/1.1
Host: example.com
```

3-

Singleton Pattern Örneği:

```
public class DatabaseConnection {
    private static DatabaseConnection instance;

    private DatabaseConnection() {
    }

    public static synchronized DatabaseConnection getInstance() {
        if (instance == null) {
            instance = new DatabaseConnection();
        }
        return instance;
    }

    public void connect() {
        System.out.println("Connected to the database.");
    }
}
```

Factory Pattern Örneği

```
public class ProductFactory {
   public Product createProduct(String name, String category, double price, int stock) {
      return new Product(name, category, price, stock);
   }
}
```

Main'de →

```
ProductFactory productFactory = new ProductFactory();

      Product product1 = productFactory.createProduct("Laptop", "Electronics", 3000, 50);
      Product product2 = productFactory.createProduct("Phone", "Electronics", 1500, 100);
      Product product3 = productFactory.createProduct("Book", "Education", 50, 200);
```

4- ECommerceApp dosyasında kodu var.

5- Password bilgisini aşağıdaki gibi hashledim.

```
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

public class UserService {

   private String encodePassword(String password) {
      try {
```

```
            MessageDigest digest = MessageDigest.getInstance("SHA-512");
            byte[] encodedHash = digest.digest(password.getBytes(StandardCharsets.UTF_8));
            StringBuilder hexString = new StringBuilder();
            for (byte b : encodedHash) {
                String hex = Integer.toHexString(0xff & b);
                if (hex.length() == 1) {
                    hexString.append('0');
                }
                hexString.append(hex);
            }
            return hexString.toString();
        } catch (NoSuchAlgorithmException e) {
            throw new RuntimeException("SHA-512 algorithm not found", e);
        }
    }

    public User registerUser(String email, String password) {
        String hashedPassword = encodePassword(password);

        User user = new User(email, hashedPassword);
        userRepository.save(user);

        return user;
    }
}
```

Sonrasında blog istatistiklerini tutmak ve hesaplamak için bir BlogStatistics modeli ve BlogService sınıfında metotlar oluşturdum.

```
public class BlogStatistics {
```

```
    private long publishedBlogCount;
    private long draftBlogCount;
    private long totalLikesCount;
    private long totalCommentCount;
}
```

Sonrasında ise "BlogService" sınıfında, aşağıda gördüğünüz gibi blog istatistiklerini hesaplayan ve blogları çeşitli kriterlere göre sıralayan metotlar oluşturdum.

```
public class BlogService {

    public void computeAndSetBlogStats(Blog blog) {
        long commentCount = blog.getComments().size();
        long publishedCount = blog.getStatus() == BlogStatus.PUBLISHED ? 1 : 0;
        long draftCount = blog.getStatus() == BlogStatus.DRAFT ? 1 : 0;
        long likeCount = blog.getLikes();

        BlogStatistics statistics = new BlogStatistics();
        statistics.setTotalCommentCount(commentCount);
        statistics.setPublishedBlogCount(publishedCount);
        statistics.setDraftBlogCount(draftCount);
        statistics.setTotalLikesCount(likeCount);

        blog.setStatistics(statistics);
    }
    public List<Blog> listBlogsByCreationDate(boolean ascending) {
        return blogRepository.findAll().stream()
                .sorted(ascending ? Comparator.comparing(Blog::getCreationDate) :
Comparator.comparing(Blog::getCreationDate).reversed())
```

```
            .collect(Collectors.toList());
    }
    public List<Blog> listBlogsByCommentCount(boolean ascending) {
        return blogRepository.findAll().stream()
            .sorted(ascending ? Comparator.comparing(blog -> blog.getComments().size()) :
Comparator.comparing(blog -> blog.getComments().size()).reversed())
            .collect(Collectors.toList());
    }
    public List<Blog> listBlogsByLikeCount(boolean ascending) {
        return blogRepository.findAll().stream()
            .sorted(ascending ? Comparator.comparing(Blog::getLikes) :
Comparator.comparing(Blog::getLikes).reversed())
            .collect(Collectors.toList());
    }
}
```