Soru 2: Creational Design Pattern'lar incelenmelidir. Örneklerle anlatınız.

Cevap: Creational Design Pattern'lar objelerin oluşturulması aşamasında kullanılan pattern'lardır.
5 adet pattern bulunmaktadır. Sırasıyla şöyle sıralanabilir:

Factory Method: Oluşturulacak olan objelerin bir interface aracılığıyla oluşturulmasını sağlar. Bu sayede birbirine benzeyen objelerin tek sınıftan türetilmesi hedeflenmektedir. Bunun için bir Factory sınıfına ihtiyaç duymaktayız.

```
class CarFactory{ //Factory Class

  public static ICar createCar(String mark, String color, double price){

     ICar car;

     if(mark.equals("BMW")){
        car = new BMW(color, price);
     }
     else if(mark.equals("Audi")){
        car = new Audi(color, price);
     }
     else{
        throw new RuntimeException(mark+" markasının üretimi desteklenmemektedir.");
     }
     return car;
  }
}
```

Objenin oluşturulacağı method içerisinde new keywordü kullanmak yerine bizim için bir 'Car' objesi ürettirmesini sağlamış olduk.

Abstract Factory: Factory method'daki dezavantaj her farklı obje için Factory class'ında değişiklik yapmamızı gerektirmesidir. Bu da 'clean code' mantığına aykırıdır. Bunu engellemek için de ürün çeşidi sayısı kadar factory oluşturmamız gerekmektedir. SOLID prensiplerinden Open/Close prensibine de uygunluk göstermektedir.

```
public interface Car {
   void honk();

}
```

```
public class Audi implements Car {

   @Override

   public void honk() {

      System.out.println("You have created Audi Car.");

   }

}
```

```
public class BMW implements Car {

    @Override
    public void honk() {
        System.out.println("You have created BMW Car.");
    }
}

public interface CarFactory {
    Car createCar();
}


public class BMWFactory implements CarFactory {
    @Override
    public Car createCar() {
        return new BMW();
    }
}


public class AudiFactory implements CarFactory {
    @Override
    public Car createCar() {
        return new Audi();
    }
}
public class Application {
    private Car car;
    public Application(CarFactory factory) {
        button = factory.createCar();
    }

    public void paint() {
        button.honk();
    }
}
```

```
public class Demo {
    private static Application configureApplication() {
        Application app;
        GUIFactory factory;
        String carBrand = “BMW”
        if (carBrand.equals("BMW")) {
            factory = new BMWFactory();
        } else {
            factory = new AudiFactory();
        }
        app = new Application(factory);
        return app;
    }

    public static void main(String[] args) {
        Application app = configureApplication();
        app.honk();
    }
}
```

Bu noktada artık yeni bir marka araç eklendiğinde eski kodlara dokunmamıza gerek kalmadan sadece configuration dosyasında değişiklik yaparak daha esnek bir yapı elde etmiş olduk.

Builder: Bu pattern'de objenin farklı özelliklerinin farklı methodlar ile ekleme yapılmaktadır. Böylece farklı objelerin aynı class'tan türemesine rağmen farklı özellikleri bulunabilmektedir.

```
public interface Builder {
    void setCarType(CarType type);
    void setSeats(int seats);
    void setEngine(Engine engine);
}
```

```
public class CarBuilder implements Builder {
    private CarType type;
    private int seats;
    private Engine engine;
```

```
    public void setCarType(CarType type) {
        this.type = type;
    }
    @Override
    public void setSeats(int seats) {
        this.seats = seats;
    }
    @Override
    public void setEngine(Engine engine) {
        this.engine = engine;
    }
    public Car getResult() {
        return new Car(type, seats, engine, transmission, tripComputer, gpsNavigator);
    }
}
```

'setter' methodlar ile aynı Car classın'dan türeyen farklı Car objeleri farklı özelliklere sahip olabilmektedir.

Prototype: Mevcut objenin klonlanmasını sağlayan bir pattern'dir. Objenin bir kopyasının doğrudan yeni bir objeye atama yaparak elde edilemediği durumlarda kullanılır (Private members).

```
public abstract class Shape {
    private int x;
    public int y;
    public String color;
    public Shape() {}
    public Shape(Shape target) {
        if (target != null) {
            this.x = target.x;
            this.y = target.y;
            this.color = target.color;
        } }
    public abstract Shape clone();
    @Override
    public boolean equals(Object object2) {
        if (!(object2 instanceof Shape)) return false;
        Shape shape2 = (Shape) object2;
        return shape2.x == x && shape2.y == y && Objects.equals(shape2.color, color);
    }
}
```

```
public class Circle extends Shape {
   public int radius;
   public Circle() { }
   public Circle(Circle target) {
      super(target);
      if (target != null) {  this.radius = target.radius;  }}
   @Override
   public Shape clone() {
      return new Circle(this);
   }
   @Override
   public boolean equals(Object object2) {
      if (!(object2 instanceof Circle) || !super.equals(object2)) return false;
      Circle shape2 = (Circle) object2;
      return shape2.radius == radius;
   }}
```

Singleton: Mevcut program akışında sadece bir kere oluşturulacak objeler için kullanılan bir pattern'dir. Uygulayabilmek için oluşturulacak class'ın constructoru (oluşturucu methodu) private yapılmalı ve objeye sadece getInstance() methodu ile erişilmelidir.

```
public final class Singleton {
   private static Singleton instance;
   public String value;


   private Singleton(String value) {
      try {
         Thread.sleep(1000);
      } catch (InterruptedException ex) {
         ex.printStackTrace();
      }
      this.value = value;
   }
   public static Singleton getInstance(String value) {
      if (instance == null) {
         instance = new Singleton(value);
      }
      return instance;
   }}
```

Soru 4: Java dünyasındaki framework'ler ve çözdükleri problemler nedir?

Cevap:

Spring: Spring Framework (Spring), Java uygulamaları geliştirmek için altyapı desteği sağlayan açık kaynaklı bir uygulama framework'üdür. En popüler Java Enterprise Edition (Java EE) çerçevelerinden biri olan Spring, geliştiricilerin old plain Java object'leri (POJO'lar) kullanarak yüksek performanslı uygulamalar oluşturmasına yardımcı olur.

Hibernate: Hibernate, veritabanıyla etkileşim kurmak için Java uygulamasının geliştirilmesini basitleştiren bir Java çerçevesidir. Açık kaynaklı, hafif, ORM (Object Relational Mapping) aracıdır. Hibernate, veri sürekliliğini korumak için JPA (Java Persistence API) özelliklerini uygular.

JSF (Java Server Faces): Component tabanlı bir MVC framework'üdür ve server tabanlı uygulamalar için yeniden kullanılabilir UI bileşenlerine sahiptir. Ana fikir, geliştiricilerin bu teknolojilerden hiçbirini derinlemesine bilmeden kullanıcı arabirimi oluşturmasına izin verecek CSS, JavaScript ve HTML gibi çeşitli istemci tarafı teknolojileri kapsüllemektir.

Struts: Apache Struts 20 yılı aşkın bir süredir piyasada ve Java EE web uygulamalarının geliştirme sürecini kolaylaştıran ve MVC mimarisini destekleyen Java Servlet API özelliklerini miras alacak şekilde tasarlandı. Struts, merkezi yapılandırmalar için XML tipi dosyalar kullanır ve web uygulamalarının genel geliştirme süresini büyük ölçüde azaltır.

Google Web Toolkit: Google Web Toolkit (kısaca GWT), web geliştiricilerinin karmaşık tarayıcı tabanlı uygulamalar oluşturmasına ve sürdürmesine yardımcı olan başka bir açık kaynaklı çerçevedir. GWT'nin amacı, geliştiricinin her tarayıcı davranışında uzman olmasını gerektirmeden web uygulamalarının verimli bir şekilde geliştirilmesini sağlamaktır.

Soru 5: Spring frameworkünün kullandığı design patternlar neler?

Cevap: Singleton, Factory Method, Proxy ve Template design patternleri kullanmaktadır.

Soru 6: SOA - Web Service - Restful Service - HTTP methods kavramlarını örneklerle açıklayınız.

Cevap: **Service Oriented Architecture** (Servis Yönelimli Mimari) kısaltmasıdır.

SOA temel olarak her hizmetin farklı birimler tarafından birbirinden bağımsız olarak çalışmasını ifade eder.

SOA yapısı kurumsal bir firmadaki farklı birimler olarak ele alınabilir.

Kurumsal bir firmada İnsan Kaynakları, Muhasebe vb. diğer birimlerden bağımsız olarak çalışır ve diğer birimlere hizmet eder.

SOA geniş kapsamı olan bir mimari yaklaşım olup çeşitli prensiplere sahiptir.

Loose Coupling

Servislerin birbirine gevşek olarak bağlı olduğunu belirtir. Böylece bir servis diğer servisten bağımsız bir şekilde çalışabilir.

Interoperability

Servislerin diğer servislerle birlikte çalışabilir olduğunu belirtir. Birlikte çalışabilirlik için ortak bir biçim kullanılır.

Reusability

Servislerin tekrar kullanılabilir olduğunu belirtir.

Abstraction

Servis iç yapısının servis kullanıcıları tarafından gizlenmesidir.

Facade

Servis ve servisi kullanan arasındaki bir bileşen/kabuk olduğunu belirtir.

Autonomy

Bir servisin diğer servislerden bağımsız olarak çalışabilir olduğunu belirtir.

Statelessness

Servislerin durumsuz olduğunu belirtir. Servislerin durum bilgisi servis isteğine göre şekil alacağını ve sürekli aynı olmadığını belirtir.

Discoverability

Servislerin keşfedilebilir olduğunu belirtir.

SOA mimarisi uygulamaya göre çeşitli prensiplere sahip olabilir.

Web Service: Web servis HTTP protokolü üzerinden diğer sistemlere/cihazlara hizmet veren yapılardır. Web servis uzak sistemler veya farklı platformlar arasında XML, JSON, CSV vb. ortak bir biçim kullanarak veri alışverişini sağlar. REST, servis yönelimli mimari üzerine oluşturulan yazılımlarda kullanılan bir veri transfer yöntemidir. HTTP üzerinde çalışır ve diğer alternatiflere göre daha basittir, minimum içerikle veri alıp gönderdiği için de daha hızlıdır. İstemci ve sunucu arasında **XML veya JSON** verilerini taşıyarak uygulamaların haberleşmesini sağlar. REST standartlarına uygun yazılan web servislerine **RESTful** servisler diyoruz.

HTTP methods: HTTP, web browser ile web server arasında iletişim kurmamızı sağlayan bir protokoldür. Başlıca method'ları şunlardır:

GET: Bu metod sunucudan veri almak için kullanılır. *GET* ve *POST* metodları en sık kullanılan metodlar olup sunucudaki kaynaklara erişmek için kullanılırlar.

POST: Bu metod ile sunucuya veri yazdırılabilir. Bu metodla istek parametreleri hem URL içinde hem de mesaj gövdesinde gönderilebilir.

PUT: Bu metod ile servis sağlayıcı üzerindeki bir kaynağı güncellenebilir. Hangi kaynak güncellenecekse o kaynağın id'sini göndermek zorunludur.

DELETE: Bu metod ile sunucudaki herhangi bir veriyi silinebilir.