**2. SDLC Nedir?**

SDLC (Software Development Life Cycle – Yazılım Geliştirme Yaşam Döngüsü)

Yazılım projelerini planlamak, yönetmek ve izlemek için kullanılan bir süreçtir. Kısa bir süre içerisinde yüksek kalitede ve düşük maliyetle yazılım üretmeyi amaçlayan bir süreçtir.

***Avantajlar***

* Yüksek düzeyde yönetim kontörlü sağlar.
* Yazılım geliştiricisinin süreç boyunca planlı ilerlemesini sağlar.
* Projede çalışan herkesin gerekli maliyet ve kaynakları anlamasını sağlar.

***Dezavantajlar***

* Proje geliştirme sürecini yavaşlatabilir.
* Projenin son aşamalarında düşük esneklik sağlar.
* Test aşaması geliştiriciler için karmaşık olabilir.

**SDLC AŞAMALARI**

1. Planlama
2. Tasarım
3. Uygulama veya Kodlama
4. Test
5. Dağıtım
6. Bakım

**SDLC Modelleri**

1. **Şelale (Waterfall Modeli)**

* En eski ve en basit olan.
* Her aşamasının kendi planı vardır.
* Eksik kalan küçük detayla süreci aksatmaya neden olur.

1. **Çevik (Agile) Modeli**

* Ürünü döngülere ayırır ve hızlı bir şekilde çalışan ürün sunar.

1. **Gelişigüzel Model:**

* Bir yöntem veya model yoktur. Kişiye özel yapılır ve gözetilmesi oldukça zordur. 1960’larda kullanılmıştır.

1. **Barok Modeli**:

* Yaşam döngü adımları temel olarak doğrusal bir şekilde gözden geçirilir. Döngü yoktur ve belgeleme ayrı bir süreç olarak ele alınır.

1. **V Süreç Modeli**

* V- model şelale modelinin gelişmiş hali olarak düşünülebilir. Bu model belirsizliklerin az olduğu, iş tanımlarının ise belirgin olduğu projelerde kullanılır.

1. **Helezonik (Spiral) Model**

* Helezonik modeli diğerlerinden ayıran özellik ise risk analizinin ön planda olması ve prototip oluşturulmasıdır. Risk analizi ön planda olduğu için hataları erken giderme imkânı tanıyabilir.

1. **Arıtımsal Geliştirme Modeli**

* Artımlı Geliştirme Modelinde proje parçalara bölünür ve kullanıcının önceliğine göre bu parçalar sıralanır. Sıralanan bu parçalar bittiğinde sırasıyla birer ara ürün geliştirilmiş olur ve bu ara ürünler de kullanıcı tarafından kullanılır. Ara ürünler her seferinde bir öncekinin üstüne bir şeyler katarak çıkartılır. Yani bu modelde bir taraftan üretim kısmı sürerken diğer tarafta ise kullanım kısmı sürer.

1. **Kodla ve Düzelt Yaşam Döngü Modeli**

* Küçük programlar için kullanılabilir, direkt ürün gerçekleştirilir ve emeklilik safhası vardır.
* Büyük projeler için kullanılamaz, bakım az vardır ama zordur, ürünü hazırlar ve kullanıma sunarsınız.

1. **Evrimsel Geliştirme Modeli**
2. **SCRUM**

* SCRUM büyük projeleri parçalara bölerek her birine “sprint” adını verir. Projeyi böler ve her bir sprinti teker teker geliştirir. SCRUM’da ekip içi iletişim çok önemlidir öyle ki her gün “SCRUM MEETINGS” denen toplantılar yapılır.

**3- Prosedürel Programlama ve Nesneye Dayalı Programlama Nedir? Hangi programlama dilleri prosedürel programlama dilidir? Hangi programlama dilleri nesneye dayalı programlama dilidir? Neden böyle bir ayrıma gidilmiştir?**

**Prosedürel Programlama:** Yazılım geliştirmeye prosedür odaklı yaklaşımıyla karakterize edilen bir programlama modelidir. Bu model programların bir dizi ardışık işlem adımından oluştuğu ve bu işlemlerin genellikle birbirini takip ettiği bir yapı içerir. Prosedürel programlama yukarıdan aşağıya yaklaşımlı bir yapıya sahiptir. Bilgisayara adım adım ne yapacağını söylersiniz.

* C
* Fortron
* COBOL
* BASIC
* Pascal
* Algol
* PL/I

Bu diller aynı zamanda prosedürel programlama dilidir.

**Nesne Yönelimli Programlama (OOP – Object Oriented Programming):** Yazılım tasarımını düzenleyen bir bilgisayar programlama modelidir. Bu yaklaşımın amacı ihtiyaç duyulan programı daha küçük parçalara bölerek yöneltilebilir ve yeniden kullanılabilir hale getirmektir. Her küçük parçanın kendine ait özelliği, verileri ve diğer küçük parçalarla nasıl iletişim kurulacağı bilgileri bulunur.

**Nesne Yönelimli Programlama Kullanılan Diller**

* Phyton
* C++
* Objective-C
* Delphi
* Swalltalk
* Java
* Swift
* C#
* Perl
* Ruby
* PHP

**Neden böyle bir ayrıma gidilmiştir?**

Prosedürel programlama ve nesne yönelimli programlama arasındaki ayrım kodun daha iyi organize edilmesi daha sürdürülebilir ve anlaşılabilir hale gelmesi daha az karmaşıklık ve daha iyi ekip çalışması sağlamak için yapılmıştır. Her iki yaklaşımın avantajları ve kullanılacakları senaryolar vardır. Bu nedenle projenin gereksinimlerine ve özelliklerine bağlı olarak tercih edilen bir model seçilir.

**4.**

|  |  |
| --- | --- |
| **Örnek Kod** | **Kullanım Amacı** |
| <script>  let tarih = new Date();  let saat = tarih.getHours();  if (saat < 19) {  alert ("iyi günler");  }  </script> | **İf** belirtilen koşulun doğru olması durumunda yürütülecek kod bloğunu belirtmek için kullanılır. |
| <script>  let tarih = new Date();  let saat = tarih.getHours();  if (saat < 19) {  alert ("iyi günler");  } else {  alert ("iyi akşamlar");  }  </script> | **else** aynı koşulun yanlış olması durumunda yürütülecek kod bloğunu belirtmek için kullanılır. |
| <script>  let sayac = 1;  while (sayac <= 5) {  console.log("Sayacın değeri: " + sayac);  sayac++;  }  </script> | **while** belirli bir koşul doğru olduğu sürece bir kod bloğunu tekrar tekrar çalıştırmak için kullanılır. |
| public class Main {  public static void main(String[] args) {  for (int i = 0; i < 5; i++) {  System.out.println(i);  }  }  } | **for** belirli bir koşul yapısının oluşması durumunda döngü sisteminin oluşmasını ve yine aynı yapı içerisinde değişken tanımlama, değer arttırma ve eksiltme gibi işlemleri yapmamıza imkan sağlayan kullanışlı bir döngü sistemidir. |
| <script>  let day = 6;  switch (day) {  case 6:  console.log("Cumartesi");  break;  case 7:  console.log("Pazar");  break;  default:  console.log("Haftaiçi");  }  </script> | **switch** ifadesi yürütülecek birçok kod bloğundan birini seçer. Genellikle bir değişkenin değerini kontrol etmek ve farklı işlemleri bu değişkenin değerine göre yapmak için kullanılır. |
|  | **case** ifadesi switch ifadesindeki bir kod bloğunu işaretler. Switch ifadesi içindeki **case** ifadeleri switch ifadesinin kontrol ettiği değişkenin veya ifadenin değeriyle eşleşen durumları tanımlar. |
|  | **break** ifadesi döngülerde veya switch ifadelerinde çalışmayı sona erdirmek için kullanılır. |
| public class Main {  public static void main(String[] args) {  int i = 0;  while (i < 10) {  if (i == 4) {  i++;  continue;  }  System.out.println(i);  i++;  }  }  } | **continue** ifadesi döngülerde veya işlem yapılarında belirli bir koşulu sağladığında döngüyü veya işlemi atlamak için kullanılır. |
| <script>  function alanHesapla(taban, yukseklik) {  let alan = taban \* yukseklik / 2;  return alan;  }  let taban = 10;  let yukseklik = 12;  let alan = alanHesapla (taban, yukseklik);  console.log("Üçgenin alanı: " + alan):  </script> | **return** bir metottan değer döndürmek veya metodu sonlandırmak için kullanılır. |
| public class Main {  int x = 9;  public static void main(String[] args) {  Main myObj = new Main();  System.out.println(myObj.x);  } } | **class** sınıf oluşturmak için kullanılır. Javada çalışan her kod satırının bir sınıf içinde olması gerekir. |
| public class PublicClass {  } | **public** sınıflar nitelikler yöntemler ve yapıcılar için kullanılan ve bunları diğer sınıflar tarafından erişilebilir hale getiren bir erişim değiştiricisidir. |
| public class Main {  private String FirstName = "Büşra";  private String LastName = "Bıçak Kopuz";    public static void main(String[] args) {  Main myObj = new Main();  System.out.println("Name: " + myObj.FirstName + " " + myObj.LastName);  }  } | **private** bir sınıfın içinde tanımlanan sınıf üyelerini diğer sınıflardan gizlemek için kullanılır. |
| class Person {  protected String email = "busrabicakk96gmail.com";  protected int age = 27;  }  class Student extends Person {  public static void main(String[] args) {  Student myObj = new Student();  System.out.println("Email: " + myObj.email);  System.out.println("Age: " + myObj.age);    }  } | **protected** nitelikler yöntemler yapıcılar için kullanılan ve onları aynı paket ve alt sınıflarda erişilebilir kılan bir erişim değiştiricisidir. |
| MyClass myObject = new MyClass(); | **new** keyword yeni nesneler oluşturur. |
| public void helloWorld () {  System.out.println(“Merhaba, Dünya”);  } | Bir methodun geriye bir değer döndürmediği durumlarda **void** kullanılır. |
| public class MyClass {  Static int staticField = 10;  } | **static** anahtar kelimesi sınıfa ait olan işlemleri ve özellikleri tanımlamak ve nesne oluşturmadan erişmek için kullanılır. |
| public class Main {  final int x = 10;  public static void main(String[] args) {  Main myObj = new Main();  System.out.println(myObj.x);  }  } | **final** anahtar kelimesi java programlama dilinde değiştirilemez veya sonlandırılabilir nesneleri sınıfları ve metotları belirtmek için kullanılır. |
| <script>  **try** {  let[] myNumbers = {1, 2, 3};  console.log(myNumbers[5]);  } **catch** (error) {  Console.log("Dizi indeksi hatası.");  }  </script> | **try** js programlama dilinde hata durumlarını ele almak ve uygun bir işlem yapmak için kullanılır. |
|  | **cath** ifadesi try bloğunda bir hata oluşması durumunda yürütülecek kod bloğunu tanımlamamıza olanak tanır. Hata durumunda çalıştırılacak kodu içerir. |
|  | **throw** ifadesi özel bir hata nesnesi oluşturmak için kullanılır. |

**5.**

|  |  |
| --- | --- |
| **Örnek Kod** | **Kullanım Amacı** |
| public class Main {  public static void main(String[] args) {  byte myNumber = 121;  System.out.println(myNumber);  }  } | **byte** anahtar kelimesi -128’den 127’ye kadar tam sayıları saklayabilen bir veri türüdür. |
| public class Main {  public static void main(String[] args) {  short myNumber = 10000;  System.out.println(myNumber);  }  } | **short** veri türü -32,768 ile 32,767 arasındaki tam sayı değerlerini depolayabilir. |
| public class Main {  public static void main(String[] args) {  int myNum = 1100000;  System.out.println(myNum);  }  } | **int** -2147483648’den 2147483647’ye kadar tam sayıları saklayabilen bir veri türüdür. |
| public class Main {  public static void main(String[] args) {  long myNum = 15000000000L;  System.out.println(myNum);  }  } | **long** -9223372036854775808 ile 9223372036854775808 arasındaki tam sayıları saklayabilen bir veri türüdür. Değer “L” ile bitirilir. |
| public class Main {  public static void main(String[] args) {  float myNumber = 5.75f;  System.out.println(myNumber);  }  } | **float** 3,4e-038 ile 3,4e+038arasındaki kesirli sayıları saklayabilen veri türüdür. Değer “f” ile bitirilir. |
| public class Main {  public static void main(String[] args) {  double myNumber = 29.99d;  System.out.println(myNumber);  }  } | **double** 1,7e-308 ile 1,7e+308 arasındaki kesirli sayıları saklayabilen bir veri türüdür. Değer “d” ile bitirilir. |
| public class Main {  public static void main(String[] args) {  char myGrade = 'B';  System.out.println(myGrade);  }  } | **char** tek ir karakteri depolamak için kullanılan bir veri türüdür. char değeri tek tırnak için alınmalıdır. |
| public class Main {  public static void main(String[] args) {  boolean isJavaFun = true;  boolean isFishTasty = false;  System.out.println(isJavaFun);  System.out.println(isFishTasty);  } } | **boolean** yalnızca doğru veya yanlış değerlerini alabilen bir veri türüdür. |
| public class Main {  public static void main(String[] args) {  String greeting = "Hello World";  System.out.println(greeting);  }  } | **String** metin verilerini temsil etmek için kullanılan bir sınıftır. |
|  |  |