**Java Interview Soru ve Cevapları-@rehberkod**

1. **Java'da "static blocks" ve "static initializers" olarak adlandırılan yapılar nelerdir?**

Classın yüklendiği veya ilk kez kullanılmaya başladığı zaman çalıştırılan kod bloklarıdır. Bu yapılar, sınıfın genel başlatma işlemlerini gerçekleştirmek için kullanılır.

"Static blocks" veya "static initializers" olarak adlandırılan bu bloklar aynı şeyi ifade eder ve sınıfın yüklendiği an veya ilk kez kullanılmadan önce çalışır. Bu bloklar, genellikle sınıfın sabitlerini, statik değişkenlerini veya diğer başlangıç işlemlerini yapmak için kullanılır. Özellikle classın ilk yüklendiği anlarda bu tür işlemler yapmak önemlidir, çünkü bu işlemler yalnızca bir kez gerçekleşmelidir.

Aşağıda, Java'da static blokların nasıl kullanılacağına dair örnek bir kod parçası bulunmaktadır:

public class MyClass {

// Static initializer block

static {

// Bu blok sınıf yüklendiğinde çalışacak kodu içerir.

System.out.println("MyClass sınıfı yüklendi.");

}

// Diğer sınıf içeriği

// ...

}

1. **Bir constructor'ı başka bir constructor'dan nasıl çağırırız?**

Bir constructor içinden başka bir constructor'ı çağırmak için this() yöntemini kullanabiliriz. Ancak bu işlemi kullanırken bazı kısıtlamalara dikkat etmemiz gerekmektedir:

this() çağrısı, constructor içindeki ilk ifade olmalıdır: this() çağrısı, constructor içindeki diğer herhangi bir işlemden veya ifadeden önce gelmelidir. Yani, constructor içinde bir başka işlemi veya ifadeyi çalıştırmadan önce this() yöntemiyle diğer bir constructor'ı çağırmanız gerekmektedir.

Bir constructor içinde birden fazla this() çağrısı kullanılamaz: Bir constructor içinde sadece bir this() çağrısı kullanabilirsiniz. Aynı constructor içinde birden fazla this() çağrısı yapılmasına izin verilmez.

Örnek:

public class MyClass {

private int value;

public MyClass() {

this(0); // Bu, diğer constructor'ı çağırır

// Başka işlemler

}

public MyClass(int value) {

this.value = value;

}

}

Yukarıdaki örnekte, this(0) ifadesi, MyClass sınıfının parametresiz constructor'ından çağrı yapar ve bu çağrı, constructor içindeki ilk ifade olarak kullanılmıştır. Bu iki kurala uyulduğunda, constructor chaining (zincirleme çağrı) işlemi sorunsuz bir şekilde çalışır ve sınıfın farklı constructor'ları arasında kod tekrarı önlenmiş olur.

1. **Java’da method overriding nedir?**

Java'da method overriding, bir alt sınıfın, üst sınıfında zaten tanımlanmış bir metodun belirli bir uygulamasını sağlamasına izin veren bir nesne yönelimli programlama kavramıdır. Bir alt sınıf, üst sınıfındaki bir metodun adı, parametre türleri (imza) ve dönüş türü ile aynı olduğunda, bu metodun üzerine yazdığı söylenir.

Java'da method overriding'i neden ve ne zaman kullanırız:

1. **Özelleştirme:** Method overriding'i kullanırsınız çünkü bir alt sınıfta belirli bir metodun özelleştirilmiş bir uygulamasını sağlamak istiyorsunuz. Örneğin, bir **Araç** üst sınıfınızda bir **motoruCalistir** metodunuz olsun, ardından **Araç** sınıfının bir alt sınıfı olan **Araba** sınıfında, **motoruCalistir** metodunu bir arabanın özgü özelliklerine göre yeniden tanımlayabilirsiniz.
2. **Polimorfizm:** Method overriding, polimorfizmin önemli bir parçasıdır. Bu, farklı alt sınıfların nesnelerini genel bir üst sınıf olarak ele almanıza olanak tanır. Bir nesne üzerinde bir metodu bir üst sınıf referansı kullanarak çağırabilirsiniz ve alt sınıfta üzerine yazılmış bir metod çalıştırılır. Bu, daha genel ve esnek kod yazmak için kullanışlıdır.

Örneğin, bir **Araç** referansı kullanarak bir **Araba** nesnesinin **motoruCalistir** metodunu çağırdığınızda, bu metodun **Araba** sınıfındaki üzerine yazılmış hali çalışır.

Aşağıda örnek bir Java kodu bulunmaktadır:

class Araç {

void motoruCalistir() {

System.out.println("Bir aracın motoru çalıştırılıyor");

}

}

class Araba extends Araç {

@Override

void motoruCalistir() {

System.out.println("Bir arabanın motoru çalıştırılıyor");

}

}

public class Ana {

public static void main(String[] args) {

Araç araç = new Araba();

araç.motoruCalistir(); // Araba sınıfındaki üzerine yazılmış metod çalışır

}

}

Yukarıdaki örnekte, **motoruCalistir** metodu **Araba** sınıfında üzerine yazılmıştır ve bir **Araç** referansı kullanılarak bu metodu çağırdığınızda, **Araba** sınıfındaki üzerine yazılmış **motoruCalistir** metodu çalışır. Bu, Java'da method overriding ve polimorfizm kavramlarını gösterir.

1. **Java’da “süper” anahtar kelimesi nedir?**

Java'da "super" anahtar kelimesi, bir alt sınıfın üst sınıfının değişkenlerine, metodlarına veya constructor’ına erişmek için kullanılan bir mekanizmadır. Bu anahtar kelime, aşağıdaki iki şekilde kullanılabilir:

1. **İlk form, üst sınıf constructor’ını çağırmak için kullanılır:** Alt sınıfın kendi constructor’ının içinde, üst sınıfın constructor’ını çağırmak için "super" anahtar kelimesi kullanılabilir. Bu, alt sınıfın constructor’ının işini tamamladıktan sonra üst sınıfın constructor’ının çalışmasını sağlar.

Örnek:

**class ÜstSınıf {**

**ÜstSınıf() {**

**System.out.println("Üst sınıfın yapıcısı");**

**}**

**}**

**class AltSınıf extends ÜstSınıf {**

**AltSınıf() {**

**super(); // Üst sınıfın constructor’ını çağır**

**System.out.println("Alt sınıfın yapıcısı");**

**}**

**}**

1. **İkinci form, üst sınıf değişkenleri ve metodlarını çağırmak için kullanılır:** Bir alt sınıf içinde, üst sınıfın değişkenlerine veya metodlarına erişmek isterseniz "super" anahtar kelimesini kullanabilirsiniz. Bu, üst sınıfın değişkenleri veya metodları alt sınıf tarafından aynı isimle aynı isimle aşırı yazıldıysa dahi üst sınıfın üyelerine erişmenizi sağlar.

Örnek:

**class ÜstSınıf {**

**int sayı = 10;**

**void metot() {**

**System.out.println("Üst sınıfın metodu");**

**}**

**}**

**class AltSınıf extends ÜstSınıf {**

**int sayı = 20;**

**void metot() {**

**System.out.println("Alt sınıfın metodu");**

**}**

**void örnekMetod() {**

**System.out.println(super.sayı); // Üst sınıfın sayısını yazdır**

**super.metot(); // Üst sınıfın metodu çağrılır**

**}**

**}**

Yani "super" anahtar kelimesi, alt sınıfın üst sınıfının üyelerine erişmesine izin verir, aynı isimdeki üyelerin üzerine yazılmasına rağmen bu üyelere erişim sağlar. "super" her zaman birinci ifade olmalıdır, yani bir yöntemde veya yapıcıda ilk kullanılan ifade olmalıdır.

1. **Java’da metod overloading ve metod overriding arasındaki farklar nelerdir?**

|  |  |
| --- | --- |
| **Metod overloading** | **Metod overriding** |
| Metot Overloading aynı class içinde gerçekleşir. | Metot Overriding iki farklı class arasında gerçekleşir, yani bir üst class (superclass) ve bir alt class (subclass) arasında. |
| Miras (inheritance) kullanımı zorunlu değildir, yani aynı sınıf içindeki metotların farklı sürümlerini oluştururken, sınıfın kendisi üzerinde çalışılır. | Miras (inheritance) kullanılır, yani bir sınıfın diğer sınıfın özelliklerini alması ve bu özellikleri değiştirmesi gereklidir. |
| Overloading metotların dönüş türü (return type) aynı olmak zorunda değildir. | Overriding metotların dönüş türü aynı olmalıdır. |
| Parametreler farklı olmalıdır, yani aynı metot adı, ancak farklı parametre listeleri kullanılır. | Parametreler aynı olmalıdır, yani aşırı yazan metot aynı isimde ve aynı parametre listesi ile yazılmalıdır. |
| Statik çok biçimlilik (static polymorphism) metot overloading ile elde edilir. Hangi metotun çağrılacağı derleme zamanında belirlenir. | Dinamik çok biçimlilik (dynamic polymorphism) metot overriding ile elde edilir. Hangi metotun çağrılacağı çalışma zamanında belirlenir. |
| Overloading sırasında bir metot diğerini gizleyemez. Yani, aynı classta overloading metotlar birbirlerini gizleyemezler. | Overriding sırasında alt classın metodu, üst classın metotunu gizler. Yani, aynı isimde ve aynı parametre listesi ile yazılmışsa, alt classın metodu üst classın metotunu gizler ve çağrıldığında alt classın metodu çalışır. |

1. **Abstract Class ve interface arasındaki farklar nelerdir?**

|  |  |
| --- | --- |
| **Interface** | **Abstract Class** |
| İnterface yalnızca soyut (abstract) metotları içerir. | Abstract sınıf, soyut metotlar, somut metotlar veya her ikisini de içerebilir. |
| İnterfacedeki metotların erişim belirleyicileri (access specifiers) her zaman public olmalıdır. | Somut (concrete) sınıfın aksine, soyut sınıfın metotlarının erişim belirleyicileri (access specifiers) daha esnek olabilir; yalnızca private kullanılamaz. |
| İnterfacede tanımlanan değişkenler her zaman public, static ve final olmalıdır. | Abstract sınıf içindeki değişkenler, private dışında diğer erişim belirleyicilerini kullanabilir. |
| Java'da çoklu kalıtımı İnterfaceler kullanılarak uygularız. | Java'da abstract sınıflarla çoklu kalıtımı başaramayız, yani bir sınıf yalnızca bir soyut sınıftan türeyebilir. |
| Bir İnterface i uygulamak için "implements" anahtar kelimesi kullanılır. | Bir soyut sınıfı uygulamak için "extends" anahtar kelimesi kullanılır. |

1. **Java için neden platformdan bağımsızdır ifadesi kullanılır?**

Java'nın platform bağımsız olmasının en önemli özelliği, Java'nın kaynak kodunun herhangi bir platform üzerinde çalıştırılabilmesidir. Diğer birçok programlama dilinde, kaynak kodu derlendikten sonra platforma özgü bir yürütülebilir dosya oluşturulur. Bu dosyalar farklı işletim sistemleri ve platformlarda çalışmayabilir. Ancak Java'da, kaynak kodu "javac" derleyicisi tarafından derlendiğinde, ".class" uzantılı yürütülebilir bir dosya oluşturulur.

Bu ".class" dosyası, Java tarafından üretilen "byte code" adı verilen bir tür ara dil kodu içerir. Byte code, sadece Java Sanal Makineleri (JVM) tarafından yorumlanabilir. Sun Microsystems (şu anda Oracle Corporation) tarafından geliştirilen JVM'ler, farklı platformlar için sunulur. Bu nedenle, bir Java programının oluşturduğu byte code Windows ortamında üretildiyse, aynı byte code Linux ortamında da çalıştırılabilir. Bu, Java'nın platform bağımsızlığını sağlar.

Özetle, Java'nın platform bağımsız olmasının nedeni, byte code'un JVM'ler aracılığıyla farklı platformlarda yorumlanabilmesidir. Bu, aynı Java programının farklı işletim sistemlerinde ve donanım platformlarında çalıştırılabilmesini mümkün kılar.

1. **Java’da overloading methodu nedir?**

Java'da metod overloading, aynı isme sahip ancak farklı parametrelerle tanımlanmış iki veya daha fazla metodu ifade eder. Java'da metodlar aynı isme sahip olabilir, ancak parametrelerin türleri, sırası veya sayısı farklı olmalıdır. Metod overloading sayesinde aynı isim altında birden fazla metot tanımlayabiliriz ve bu metotlar farklı işlevler veya girdilerle çalışabilir.

Java'da static polymorphism, yani hangi metodu çağıracağınızın derleme zamanında belirlenmesi, metod overloading ile elde edilir. Derleyici, overloading metotların ismini, parametre sayısını ve parametre türlerini dikkate alarak hangi metodu çağıracağınızı belirler. Bu nedenle, aynı metot ismi altında birden fazla metot tanımlanabilir ve bu metotlar overload edilmiş olur.

Önemli bir not olarak, metotların dönüş türü (return type) metot signature ın bir parçası değildir. Yani, farklı dönüş türlerine sahip metotlar overload edilebilir, ancak bu dönüş türleri metodu çağırmak için yeterli değildir. Hangi metodu çağıracağınızı belirlemek için derleyici, metot ismi ve parametrelerin türleri ve sırası gibi faktörleri kullanır. Bu nedenle, overload edilmiş metotlar aynı ismi paylaşsa bile farklı parametrelerle çağrılabilir.

1. **JIT (Just-In-Time) derleyiciyisi nedir?**

JIT derleyici, Java bytecode'unu çalıştırılabilir kod haline getiren bir bileşen olarak JVM içinde bulunur. Ancak, JIT derleyici bir Java programını tamamen çalıştırılabilir kod haline dönüştürmez. Bunun yerine, Java programı çalıştırıldığında ihtiyaç duyulan kısımları bytecode'dan çalıştırılabilir kod haline çevirir. Böylece, Java programı daha etkili bir şekilde çalıştırılabilir.

1. **Java’da bytecode nedir?**

Bytecode, bir Java programının kaynak kodunun derlenmesi sonucunda oluşturulan .class dosyasında bulunan bir dizi talimat setidir. Bytecode, bir makine bağımsız dil olarak kabul edilir ve yalnızca Java Sanal Makinesi (JVM) tarafından yürütülmek üzere tasarlanmış talimatları içerir. Yani, Java kaynak kodu derlendikten sonra oluşturulan bytecode, JVM tarafından yorumlanabilir ve çalıştırılabilir. Bu, Java'nın platform bağımsızlığına katkıda bulunan önemli bir özelliktir, çünkü bytecode aynıdır, ancak farklı işletim sistemlerinde çalışan JVM'ler tarafından yürütülür.

1. **Java’da this() ve süper() keywords leri arasındaki farklar nelerdir?**

this():

this() anahtar kelimesi, aynı sınıf içerisindeki bir başka yapıyı (genellikle bir başka constructor'ı) çağırmak için kullanılır.

Genellikle overloading constructor'ları çağırmak veya aynı sınıf içindeki diğer constructor'larda tekrarlayan kodu önlemek için kullanılır.

this() anahtar kelimesi, aynı sınıf içindeki başka bir constructor'ı çağırmadan önce kullanılır ve constructor içinde sadece bir kere çağrılabilir.

İlgili constructor çağrıldıktan sonra bu constructor içindeki diğer işlemler devam eder.

super():

super() anahtar kelimesi, alt sınıfın constructor'ı içinde üst sınıfın constructor'ını çağırmak için kullanılır.

Java'da her alt sınıfın bir üst sınıfı vardır, ve bu nedenle alt sınıf constructor'ları, üst sınıf constructor'ını çağırmalıdır.

super() anahtar kelimesi, alt sınıf constructor'ının ilk satırında kullanılmalıdır.

1. **Java’da class kavramı nedir?**

Bir class, Nesne Yönelimli Programlama (OOP) içinde temel birimlerden biridir.

Bir class, nesnelerin tasarım şablonunu veya yapısını temsil eder.

Classlar, değişkenler (variables) ve metotları (methods) tanımlarlar.

Her class, belirli bir türde nesnelerin oluşturulabileceğini tanımlar. Örneğin, "Department" classı, departman türünde nesnelerin oluşturulabileceğini belirtir.

Java'da tüm program yapısı classlar içinde tanımlanır.

Bir Java uygulamasında en az bir class ve bir "main" metodu (main method) bulunmalıdır.

Bir class tanımı "class" anahtar kelimesi ile başlar.

Class tanımı, aynı isme sahip bir class dosyasında (".java" uzantılı) saklanmalıdır.

Class dosyalarının adı, classın adı ile aynı olmalıdır.

Class dosyaları derlendiğinde JVM (Java Sanal Makinesi), classı yükler ve bir ".class" dosyası oluşturur.

Program çalıştırıldığında, classı çalıştırır ve "main" metodu çalıştırır.

Yani, classlar, nesnelerin oluşturulmasını ve programların yapılandırılmasını tanımlamak için kullanılan temel yapı taşlarıdır.

1. **Java’da object nedir?**

Bir object, bir classın örneğidir. Yani, class bir türün (type) tasarımını veya şablonunu belirtirken, objectler bu türün somut örnekleridir.

Her object, bir classa aittir. Class, objectlerin temel yapısını ve davranışını tanımlar.

Bir object, iki temel bileşeni içerir: durum (state) ve davranış (behavior).

Durum, objectin özelliklerinin veya niteliklerinin değerlerini temsil eder. Bu özellikler, classın tanımladığı öznitelikler veya değişkenler olabilir.

Davranış, objectin yapabileceği işlemleri veya yöntemleri temsil eder. Bu yöntemler, classın tanımladığı işlevleri ifade eder.

Objectler ayrıca "örnek"(instance) olarak da adlandırılır.

Bir classın objectini oluşturmak için, new anahtar kelimesini kullanarak bir örnek (instance) oluşturulur. Örneğin, FirstClass classının bir örneğini oluşturmak için aşağıdaki gibi bir kod kullanılabilir:

FirstClass f = new FirstClass();

Burada f, FirstClass classının bir örneğini temsil eder ve bu örneğe erişmek için kullanılır.

Yani, bir object, bir classın belirlediği yapı ve davranışa sahip somut bir örneğidir. Objectler, classların özelliklerini ve işlevlerini kullanarak programın temel taşlarıdır ve programlar objectler aracılığıyla veri ve işlemler üzerinde çalışırlar.

1. **Method nedir?**

Java'da bir "metot" (method), bir class içindeki işlevsel bir bloktur ve belirli bir nesne üzerinde çalıştırılabilir. İşte

Metotlar, bir class içindeki işlevleri veya operasyonları tanımlarlar. Bir classın nesneleri, bu metotları çağırarak belirli işlemleri gerçekleştirebilir.

Bir metot, şu temel unsurları içerir:

Metot adı: Metodun adı, metodu tanımlayan bir isimdir. Bu isimle metodu çağırabilirsiniz.

Parametreler veya argümanlar: Metot, çalıştırılması sırasında aldığı verileri işlemek için parametreleri kullanabilir. Parametreler metot adının hemen sonra parantez içinde belirtilir.

Dönüş türü: Metot, bir sonuç döndürebilir veya dönüş yapmayabilir. Dönüş türü, metotun ne tür bir değer dönebileceğini belirtir. Örneğin, int veya float gibi bir veri türü olabilir.

Metotların temel sözdizimi şu şekildedir:

dönüş\_türü metot\_adı(parametre\_listesi) {

// Metodun işlevselliğini tanımlayan kod burada bulunur

}

Örnek bir metot tanımı:

public float toplama(int a, int b, int c) {

int sonuc = a + b + c;

return sonuc;

}

Metotlar birden fazla parametre alabilir ve parametreler virgülle ayrılır. Metotun işlevselliğini tanımlayan kod, süslü parantezlerin içine yerleştirilir.

Sonuç olarak, bir metot Java programında bir işlemi veya işlevselliği temsil eder. Bu işlevselliği çağırmak için metot adını ve gerekli parametreleri kullanabilirsiniz. Metotlar, programlarınızın düzenli ve modüler olmasına yardımcı olan önemli yapı taşlarıdır.

1. **Encapsulation nedir?**

Encapsulation, verileri bir tekil birim olan class içine sarmak veya yerleştirmek ve verilerin yanlış kullanımından korumak işlemidir.

Encapsulation (kapsülleme), verileri ve bu verilere erişim yöntemlerini (metotları) bir class içinde toplama ve düzenleme sürecini ifade eder. Bu, verilerin dışarıdan doğrudan erişilmesini sınırlar ve kontrol altına alır.

Java'da encapsulation, verilere erişim kontrolü ile desteklenir. Bu kontrol, dört erişim kontrol belirleyicisi (modifier) kullanılarak sağlanır: public, private, protected ve default (veya package-private).

Bir örnek üzerinden açıklamak gerekirse, bir "araba" classını ele alalım. Bir arabanın içinde birçok parça ve veri bulunur, ancak sürücünün sadece arabayı nasıl çalıştıracağı ve nasıl durduracağı hakkında bilgiye ihtiyacı vardır. Bu nedenle, encapsulation kullanarak sürücüye sadece gerekli olan bilgileri sunabilir ve geri kalan bilgileri gizleyebiliriz.

Encapsulation, verilere erişimi kontrol altına alarak veri güvenliğini artırır ve bir classın iç yapısını gizleyerek daha sade ve anlaşılır bir arabirim sunar.

Sonuç olarak, encapsulation (kapsülleme), verilerin gizliliğini korumak ve erişimi kontrol altına almak için kullanılan bir nesne yönelimli programlama prensibidir. Bu prensip, daha güvenli ve modüler yazılım geliştirmeye yardımcı olur.

1. **Java'da main methodu neden public, static ve void olarak tanımlanmıştır?**

public: public bir erişim belirleyicisidir ve bu, main methodunun diğer sınıflardan (veya sınıf dışından) erişilebilir olmasını sağlar. Java programlarının başlangıç noktası olan main methodu, JVM (Java Virtual Machine) tarafından çağrılmalıdır, bu nedenle public olarak işaretlenmesi, başka sınıflardan veya JVM'den erişilebilir olmasını sağlar.

static: main methodu nesne oluşturmadan çağrılabilmesi gereken özel bir methoddur. Java programının başlangıcı için kullanıldığı için bu method, sınıf düzeyinde (yani sınıfın bir örneği oluşturulmadan) çağrılmalıdır. Bu nedenle static olarak işaretlenir. Bir method static olarak işaretlendiğinde, bu method sınıfın bir örneği (nesnesi) oluşturulmadan çağrılabilir.

void: main methodu bir sonuç döndürmez. Java'da methodlar belirli bir türde bir değer dönebilirler, ancak main methodu programın başlangıcı olduğu için herhangi bir değer döndürmesine gerek yoktur. Bu nedenle void olarak işaretlenir. Yani, main methodu işlemleri gerçekleştirir, ancak herhangi bir değer geri döndürmez.

Özetle, public static void main(String[] args) ifadesi, Java programlarının başlangıcı için özel olarak tanımlanan main methodunun imzasını belirtir ve bu imza, programın başlatılmasını ve çalıştırılmasını sağlar. Bu nedenle, main methodu programın giriş noktasıdır ve yukarıdaki imza, Java'nın bu noktayı bulmasına ve çalıştırmasına yardımcı olur.

1. **Java’da constructor nedir?**

Java'da bir constructor, bir sınıfın örneklerini oluştururken bu örneklerin başlangıç değerlerini ayarlamak için kullanılan özel bir metoddur. İnşa edici metodlar, bir sınıfın yapısını ve davranışlarını belirlemek için kullanılan bir tür yapıcı işlevi görürler. İnşa ediciler, yeni bir nesne oluşturulduğunda otomatik olarak çağrılır ve nesnenin başlangıç durumunu ayarlamak için kullanılır.

Java'da iki temel türde constructor bulunur:

Default Constructor (Parametresiz Constructor): Bir sınıfın herhangi bir constructor tanımlanmadığında, Java otomatik olarak bir varsayılan constructor sağlar. Bu varsayılan constructor, sınıfın nesnelerini oluştururken herhangi bir argüman almayan ve genellikle sınıfın içinde herhangi bir işlem yapmayan bir constructor'dır. Örnek olarak:

public class MyClass {

public MyClass() {

// Default constructor

}

}

Parameterized Constructor (Parametre Alan Constructor): Parametreli constructorlar, nesneleri oluştururken belirli parametreleri alabilir ve bu parametrelere dayalı olarak nesnenin başlangıç durumunu ayarlar. Bu tür constructorlar genellikle sınıfın özelliklerini başlangıç değerleri ile doldurmak için kullanılır. Örnek olarak:

public class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

}

Yukarıdaki örnek, Person sınıfı için bir parametreli constructor gösterir. Bu constructor, name ve age adlı iki parametre alır ve bu parametreleri kullanarak Person nesnesinin başlangıç değerlerini ayarlar.

Constructorlar, sınıfın yapısını tanımlamak ve sınıf örneklerini başlatmak için önemli bir rol oynarlar ve Java'da sınıf oluştururken yaygın olarak kullanılırlar.

1. **Java'da length ve length() ifadeleri arasındaki fark nedir?**

length(): Bu, bir String nesnesinin metin uzunluğunu döndüren bir metoddur. length() bir metin nesnesinin karakterlerinin sayısını verir. Örnek olarak:

String str = "Hello World";

int uzunluk = str.length(); // uzunluk, "Hello World" ifadesinin karakter sayısı olan 11'i döndürecektir.

length() yöntemi bir String nesnesi üzerinde çağrılır ve bu nesnenin uzunluğunu karakter sayısı olarak döndürür.

length: Bu, dizilerin veya dizilimlerin (arrays) uzunluğunu döndüren bir özelliktir. Dizilerin boyutunu (kaç eleman içerdiğini) belirtir. Örneğin:

String[] days = {"Sun", "Mon", "Wed", "Thu", "Fri", "Sat"};

int uzunluk = days.length; // uzunluk, "days" dizisinin içerdiği eleman sayısı olan 6'yı döndürecektir.

length özelliği, bir dizi veya dizilim üzerinde kullanılır ve bu yapıdaki elemanların sayısını verir.

Özetle, length() bir String nesnesinin karakter sayısını döndürürken, length özelliği bir dizi veya dizilimin içerdiği eleman sayısını döndürür. Bu iki ifade arasındaki fark, kullanıldıkları veri türüne ve kullanım bağlamına dayanır.

1. **ASCII (American Standard Code for Information Interchange) kodu nedir?**

ASCII, bilgisayarların ve iletişim sistemlerinin metin karakterlerini temsil etmek için kullanılan bir karakter kodlaması standardıdır. ASCII, 0'dan 127'ye kadar olan karakterleri tanımlayan yedi bitlik bir karakter setidir. 0 ile 127 arasındaki karakterler, İngilizce alfabesi, rakamlar, temel noktalama işaretleri ve bazı kontrol karakterlerini içerir.

Bu standart, özellikle eski bilgisayar sistemlerinde ve yazılım uygulamalarında yaygın olarak kullanılmıştır. ASCII karakter seti, İngilizce dilindeki metinlerin temsilinde işe yarar, ancak diğer dillerin karakterlerini veya sembollerini desteklemez. Bu nedenle, ASCII karakter seti sınırlı bir karakter yelpazesine sahiptir ve yalnızca temel İngilizce karakterleri içerir.

Açıklamada belirtildiği gibi, C programlama dili geleneksel olarak ASCII karakterlerini kullanır. Bu nedenle, C dilinde yazılmış programlar genellikle İngilizce karakterlerle yazılmalıdır. Başka dillerdeki karakterlerin veya sembollerin C programlarına dahil edilmesi zordur, çünkü ASCII karakter seti bu karakterleri desteklemez.

Günümüzde, daha geniş karakter kümelerini ve uluslararası dil desteğini sağlayan Unicode gibi karakter kodlamaları, ASCII'nin yerini almıştır. Unicode, dünya genelinde farklı dillerdeki karakterlerin ve sembollerin temsilini sağlar ve çok daha geniş bir karakter yelpazesi sunar.

1. **Unicode nedir?**

Unicode, dünya genelinde tüm dillerin karakterlerini ve sembollerini temsil etmek amacıyla oluşturulan bir karakter kümesidir. Unicode, Unicode Consortium adlı bir kuruluş tarafından geliştirilmiştir ve farklı dillerdeki karakterlerin ve sembollerin benzersiz bir kodlanmış temsilini sağlar.

Aşağıdaki noktalara dikkat edilmelidir:

Unicode Karakter Aralığı: Unicode karakterleri 16 bit (2 byte) olarak temsil edilir, bu da toplamda 65,536 (0'dan 65,535'e kadar) farklı karakteri içeren bir karakter kümesi sağlar. Bu, dünya genelindeki birçok dilin karakterlerini içerir.

Java ve Unicode: Java, Unicode karakterlerini destekler. Java programlama dilindeki String'ler, tanımlayıcılar (identifiers) ve yorumlar (comments) gibi metin verileri için Unicode kullanılır. Bu, Java'nın farklı dillerdeki karakterleri ve sembolleri doğru bir şekilde temsil etmesini sağlar. Örneğin, Java'da Telugu dilindeki karakterler kullanılarak tanımlayıcılar (değişken adları) oluşturulabilir ve Telugu dilinde yorumlar yazılabilir.

Java'nın Unicode'ı desteklemesi, çok dilli uygulamalar geliştirmek için kullanışlıdır. Java, uluslararası yazılım geliştirme için yaygın olarak tercih edilen bir dildir çünkü Unicode desteği sayesinde dünya genelinde kullanılabilir ve çoklu dil desteği sağlar.

1. **Java'da '>>' ve '>>>' operatörleri arasındaki fark nedir?**

'>>' (Sağa Kaydırma Operatörü):

'>>' operatörü, bir değerin tüm bitlerini belirtilen sayıda sağa kaydırmak için kullanılır.

Sağa kaydırma işlemi yapılırken, en sol taraftaki bitin değeri önceki değerine göre korunur ve sağa doğru belirtilen sayıda boşluk bırakılır.

Eğer kaydırma işlemi sonucunda boşalan bitler, kaydırma işlemi yapılan sayının işareti ile doldurulur. Yani, eğer işaretli bir tamsayıyı kaydırıyorsanız, sol taraftaki boşluklar işaret biti (negatiflik) ile doldurulur.

Örnek:

int a = 15;

a = a >> 3; // Sonuç: 1

Bu örnekte, 15 değeri ikili olarak '00001111' temsil edilir ve bu değer 3 kez sağa kaydırıldığında '00000001' elde edilir.

'>>>' (Sağa Kaydırma Operatörü - İşaretsiz):

'>>>' operatörü, sağa kaydırma işlemi yaparken işaretsiz (unsigned) bir kaydırma gerçekleştirir.

Bu operatör ile kaydırma işlemi sonucunda boşalan bitler her zaman sıfır ile doldurulur. Yani, işaret biti (negatiflik) göz ardı edilir.

Örnek:

int a = -15;

a = a >>> 3; // Sonuç: 536870909

Bu örnekte, -15 değeri ikili olarak '11111111111111111111111111110001' temsil edilir. Ancak '>>>' operatörü ile 3 kez sağa kaydırıldığında, işaret biti dikkate alınmaz ve sonuç '00011111111111111111111111111100' olur. Bu nedenle sonuç işaretsiz bir tam sayı olarak 536870909'dur.

'>>' ve '>>>' operatörleri arasındaki temel fark, işaret bitinin (negatiflik) nasıl işlendiği ve kaydırma işlemi sonucunda boşalan bitlerin nasıl doldurulduğudur.

1. **Java kodlama standartları veya Java kodlama kurallarını (Java Coding Conventions) açıklayınız?**

Java kodlama standartları veya Java kodlama kuralları (Java Coding Conventions) genellikle yazılım geliştiricilerin kodlarını daha okunaklı ve tutarlı bir şekilde yazmalarına yardımcı olur. Java sınıfları için kodlama standartlarına dair önemli noktalar:

Sınıf İsimleri Büyük Harfle Başlamalıdır:

Java sınıf isimleri büyük harfle başlamalıdır. Bu, sınıfları diğer öğelerden (değişkenler, metotlar vb.) ayırt etmek için kullanışlıdır.

Örnek: Çalışan veya MüşteriHesabı

Sınıf İsimleri İsimler Olmalıdır:

Sınıf isimleri genellikle isim (noun) olmalıdır ve sınıfın temsil ettiği nesnenin ne olduğunu açıkça yansıtmalıdır.

Örnek: Çalışan sınıfı bir çalışanı temsil ediyorsa, ismi Çalışan olabilir.

Birden Fazla Kelimeden Oluşan Sınıf İsimleri:

Eğer bir sınıf adı birden fazla kelimeden oluşuyorsa, iç kelimelerin ilk harfi büyük olmalıdır. Buna "Camel Case" denir.

Örnek: MüşteriHesabı, KişiProfilBilgisi, Kutuİşlemi

Java kodlama standartları, kodun okunabilirliğini artırmak, yazılım projelerini daha kolay yönetilebilir hale getirmek ve geliştiriciler arasında tutarlılık sağlamak amacıyla kullanılır. Bu kurallara uymak, kodun daha iyi anlaşılmasını ve bakımını kolaylaştırır ve daha profesyonel bir yazılım geliştirme süreci sunar.

1. **Java’da interfaceler için standartlar nelerdir?**

İnterface İsmi Büyük Harfle Başlamalıdır:

Java interface isimleri büyük harfle başlamalıdır. Bu, interfaceleri diğer öğelerden (sınıflar, değişkenler vb.) ayırt etmek için kullanışlıdır.

Örnek: Çalıştırılabilir, Serileştirilebilir

İnterface İsimleri Sıfat Olmalıdır:

İnterface isimleri genellikle sıfat (adjective) olmalıdır ve interface in temsil ettiği yeteneği, özelliği veya davranışı açıkça yansıtmalıdır.

Örnek: ÇoğulUyumluluk interface i, bir sınıfın çoğul uyumlu (Runnable) olduğunu ifade eder. Serileştirilebilir interface i, bir sınıfın serileştirilebilir (Serializable) olduğunu ifade eder. Klonlanabilir interface i, bir sınıfın klonlanabilir (Cloneable) olduğunu ifade eder.

Java'da bu tür kodlama standartları ve adlandırma kuralları, kodun okunabilirliğini artırır ve yazılım projelerini daha anlaşılır ve yönetilebilir hale getirir. Ayrıca, diğer geliştiriciler için daha açık ve tutarlı bir programlama deneyimi sağlar. Bu standartlara uyum, yazılım geliştirme sürecini daha profesyonel hale getirir ve işbirliğini kolaylaştırır.

1. **Java’da metod kullanım standartları nelerdir?**

Metot İsimleri Küçük Harfle Başlamalıdır:

Java metot isimleri küçük harfle başlamalıdır. Bu, metotları diğer öğelerden (sınıflar, arayüzler, değişkenler vb.) ayırt etmek için kullanışlıdır.

Örnek: toString(), hesapla().

Metot İsimleri Genellikle Fiil Olmalıdır:

Metot isimleri genellikle fiil (verb) olmalıdır çünkü metotlar bir eylemi gerçekleştirir.

Örnek: hesapla(), oluştur(), güncelle().

Birden Fazla Kelimeden Oluşan Metot İsimleri:

Eğer bir metot adı birden fazla kelimeden oluşuyorsa, iç kelimelerin ilk harfi büyük olmalıdır. Buna "Camel Case" denir.

Örnek: getArabaAdı(), getArabaNumarası(). Bu şekilde metot adları daha okunaklı ve anlaşılır olur.

Metot İsmi Genellikle Fiil ve İsimin Kombinasyonu Olmalıdır:

Metot ismi, metotun ne yaptığını ve neyi döndürdüğünü açıkça ifade etmelidir. Genellikle bir eylemi (fiil) ve bu eylemin üzerinde çalıştığı nesneyi (isim) içerir.

Örnek: getArabaAdı() metodu, bir arabanın adını döndürür. hesaplaToplam() metodu, bir toplamı hesaplar ve sonucu döndürür.

Java'da bu tür kodlama standartları ve adlandırma kuralları, kodun okunabilirliğini artırır ve yazılım projelerini daha anlaşılır ve yönetilebilir hale getirir. Ayrıca, diğer geliştiriciler için daha açık ve tutarlı bir programlama deneyimi sağlar. Bu standartlara uyum, yazılım geliştirme sürecini daha profesyonel hale getirir ve işbirliğini kolaylaştırır.

1. **Java’da variable(değişken) kullanım standartları nelerdir?**

Değişken İsimleri Küçük Harfle Başlamalıdır:

Java değişken isimleri küçük harfle başlamalıdır. Bu, değişkenleri diğer öğelerden (sınıflar, metotlar, arayüzler vb.) ayırt etmek için kullanışlıdır.

Örnek: string, değer, çalışanAdı, çalışanMaaşı.

Değişken İsimleri Genellikle İsim Olmalıdır:

Değişken isimleri genellikle isim (noun) olmalıdır ve değişkenin temsil ettiği verinin ne olduğunu açıkça yansıtmalıdır.

Örnek: sayı, müşteri, kitapAdı, öğrenciNotları.

Kısa ve Anlamlı İsimler Tavsiye Edilir:

Değişken isimleri kısa olmalıdır, ancak anlamlı olmalıdır. Değişkenin temsil ettiği veriyi açıkça ifade etmelidir.

Örnek: yaş, fiyat, ad, boyut.

Birden Fazla Kelimeden Oluşan Değişken İsimleri:

Eğer bir değişken adı birden fazla kelimeden oluşuyorsa, iç kelimelerin ilk harfi büyük olmalıdır. Buna "Camel Case" denir.

Örnek: çalışanAdı, öğrenciNotları, kitapListesi.

Java'da bu tür kodlama standartları ve adlandırma kuralları, kodun okunabilirliğini artırır ve yazılım projelerini daha anlaşılır ve yönetilebilir hale getirir. Ayrıca, diğer geliştiriciler için daha açık ve tutarlı bir programlama deneyimi sağlar. Bu standartlara uyum, yazılım geliştirme sürecini daha profesyonel hale getirir ve işbirliğini kolaylaştırır.

1. **Java’da sabitler (constant) ile ilgili standartlar nelerdir?**

Sabitler yalnızca büyük harflerle yazılmalıdır.

Örneğin:

public static final int MAX\_VALUE = 100;

public static final String DATABASE\_NAME = "MyDatabase";

Sabit isimleri iki kelimenin birleşiminden oluşuyorsa, bu kelimeler alt çizgi (\_) ile ayrılmalıdır. Bu, sabitlerin okunabilirliğini artırır.

Örneğin:

public static final int MAX\_LENGTH = 255;

public static final String API\_KEY = "my\_api\_key";

Sabit isimleri genellikle isimler veya nesnelerle ilişkilendirilen kelimeler olmalıdır. Bu, sabitin ne tür bir veriyi temsil ettiğini açıkça belirtir.

Örneğin:

public static final int DEFAULT\_TIMEOUT = 5000;

public static final String ERROR\_MESSAGE = "An error occurred.";

Bu kurallar, sabitlerin kodunuzda daha iyi anlaşılabilir ve bakımı daha kolay olmasını sağlar. Ayrıca, sabitlerin değerlerinin değiştirilemez (final) olduğundan emin olur, böylece bu değerlerin yanlışlıkla değiştirilmesini engeller ve kodunuzun güvenilirliğini artırır.

1. **Java'da "IS-A" ilişkisi nedir?**

Java'da "IS-A" ilişkisi, genellikle miras (inheritance) olarak bilinir. "IS-A" ilişkisi, bir sınıfın başka bir sınıftan türetildiği veya kalıtım aldığı durumu ifade eder. Bu türetilen sınıf, türeten sınıfın özelliklerini ve davranışlarını devralır. Java'da "IS-A" ilişkisi, "extends" anahtar kelimesi kullanılarak gerçekleştirilir. Miras alınan sınıf, miras alan sınıfın bir alt sınıfıdır.

Örnek olarak, "Araç" sınıfını ele alalım:

public class Araç {

// Araç sınıfına ait özellikler ve davranışlar burada tanımlanır.

}

Bu durumda, "Araç" sınıfı bir üst sınıftır. Şimdi bu sınıftan türetilen "Motosiklet" ve "Araba" sınıflarını düşünelim:

public class Motosiklet extends Araç {

// Motosiklet sınıfının özellikleri ve davranışları burada tanımlanır.

}

public class Araba extends Araç {

// Araba sınıfının özellikleri ve davranışları burada tanımlanır.

}

Bu durumda, "Motosiklet" ve "Araba" sınıfları "Araç" sınıfından türetilmiştir ve "Araç" sınıfının özelliklerini ve davranışlarını miras almışlardır. Bu, "Motosiklet bir Araç'tır" ve "Araba bir Araç'tır" ifadelerini doğru bir şekilde ifade eder.

Miras (inheritance), kodun yeniden kullanılabilirliğini artırır ve sınıf hiyerarşileri oluşturarak nesne yönelimli programlamada (OOP) önemli bir konsepttir.

1. **Java'da "HAS-A" ilişkisi nedir?**

Java'da "HAS-A" ilişkisi, "kompozisyon" veya "birleştirme (aggregation)" olarak da bilinir. "HAS-A" ilişkisi, bir sınıfın başka bir sınıfı içerdiği veya ona sahip olduğu bir ilişkiyi ifade eder. Bu ilişki, bir sınıfın başka bir sınıfın nesnesini içermesini veya bu nesneye erişim sağlamasını içerir. "HAS-A" ilişkisi, Java'da "extends" anahtar kelimesi gibi özel bir anahtar kelime kullanılmadan gerçekleştirilir.

Örnek olarak, "Araba" sınıfının içinde bir "Motor" sınıfına sahip olduğunu düşünelim:

public class Motor {

// Motor sınıfına ait özellikler ve davranışlar burada tanımlanır.

}

public class Araba {

private Motor motor; // Araba sınıfı, Motor sınıfının bir nesnesini içerir.

public Araba() {

motor = new Motor(); // Araba nesnesi oluşturulduğunda bir Motor nesnesi de oluşturulur.

}

// Araba sınıfına ait diğer özellikler ve davranışlar burada tanımlanır.

}

Bu durumda, "Araba" sınıfı bir "Motor" nesnesini içeriyor. Bu, "Araba HAS-A Motor" ilişkisini ifade eder. "Araba" sınıfı, "Motor" sınıfının özelliklerine ve davranışlarına erişebilir ve onları kullanabilir.

"HAS-A" ilişkisi, kodun yeniden kullanılabilirliğini artırır, çünkü bir sınıfın başka bir sınıfın özelliklerini veya davranışlarını içermesi sayesinde, kod tekrarı önlenir ve kodun daha modüler ve anlaşılır hale gelmesine yardımcı olur. Bu tür ilişkiler, nesne yönelimli programlamanın (OOP) temel kavramlarından biridir.

1. **Java'da "IS-A" ilişkisi ile "HAS-A" ilişkisi arasındaki farklar nelerdir?**

|  |  |
| --- | --- |
| "IS-A" ilişkisi | "HAS-A" ilişkisi |
| Miras (inheritance) ilişkisi olarak bilinir. | Kompozisyon (composition) veya birleştirme (aggregation) ilişkisi olarak bilinir. |
| Bir sınıfın başka bir sınıftan türetildiği veya kalıtım aldığı bir ilişkiyi ifade eder. | Bir sınıfın başka bir sınıfı içerdiği veya ona sahip olduğu bir ilişkiyi ifade eder. |
| "extends" anahtar kelimesi kullanılarak gerçekleştirilir. Türetilen sınıf, türeten sınıfın bir alt sınıfıdır. | Özel bir anahtar kelime kullanılmadan gerçekleştirilir. Sınıf içinde diğer bir sınıfın nesnesi saklanarak veya bu nesneye erişilerek gerçekleştirilir. |
| Örnek: "Araba bir Taşıt'tır." ("Car is a Vehicle.") ifadesi doğru bir IS-A ilişkisini ifade eder. | Örnek: "Araba bir Motor'a sahiptir." ("Car has an Engine.") ifadesi doğru bir HAS-A ilişkisini ifade eder. |
| Mirasın başlıca avantajı kodun yeniden kullanılabilirliğini artırmaktır. | Burada dikkat edilmesi gereken önemli bir nokta, HAS-A ilişkisi ile türetilen sınıfın ayrı bir sınıf olduğudur. Yani Araba sınıfı Motor sınıfının bir alt sınıfı değil, yalnızca bir Motor nesnesine sahiptir. |

1. **Java’da instanceof operatörünü açıklayınız?**

instanceof operatörü, bir nesnenin hangi türde olduğunu test etmek için kullanılır. Bu operatör, bir referans ifadesini ve bir hedef türü alır ve referans ifadesinin belirtilen hedef türün bir alt türü olup olmadığını kontrol eder. instanceof ifadesi doğru (true) veya yanlış (false) bir değer döndürür.

instanceof operatörünün sözdizimi şu şekildedir:

<referans ifadesi> instanceof <hedef tür>

instanceof, aşağıdaki iki sonucu üretebilir:

true: Eğer referans ifadesi, hedef türün bir alt türü ise, instanceof operatörü true değeri döndürür.

false: Eğer referans ifadesi hedef türün bir alt türü değilse veya referans ifadesi null ise, instanceof operatörü false değeri döndürür.

Örnek olarak, aşağıdaki Java kodu instanceof operatörünü kullanır:

public class InstanceOfOrnegi {

public static void main(String[] args) {

Integer a = new Integer(5);

if (a instanceof java.lang.Integer) {

System.out.println(true);

} else {

System.out.println(false);

}

}

}

Bu örnekte, a değişkeni bir Integer türündedir ve a instanceof java.lang.Integer ifadesi true değeri döndürür çünkü a, java.lang.Integer türünün bir örneğidir.

instanceof operatörü, nesnelerin türlerini kontrol etmek, uygun işlemleri gerçekleştirmek ve tip güvenliğini sağlamak için kullanılır. Eğer bir referansın belirli bir türün bir alt türü olup olmadığını kontrol etmeniz gerekiyorsa, instanceof operatörü bu işlemi yapmanıza yardımcı olur.