for Döngüleri

for döngüleri aslında while döngülerinin daha genel bir biçimidir. for döngü deyiminin genel biçimi şöyledir:

for ([ifade1]; [ifade2]; [ifade3])
	<deyim>

for döngüsünün ikinci kısmındaki ifade boolean türden olmak zorundadır. Birinci ve üçüncü kısımdaki ifadeler herhangi bir türden olabilir. İki noktalı virgül her zaman parantez içerisinde bulunmak zorundadır. Fakat ifade1, ifade2 ve ifade3 bulunmak zorunda değildir. for döngüsü şöyle çalışır: Döngüye girişte birinci kısımdaki ifade bir kez yapılır. Bir daha yapılmaz. Döngünün yinelenmesinden ikinci kısımdaki ifade sorumludur. Döngü bu ifade true olduğu sürece yinelenir. Üçüncü kısımdaki ifade her döngü deyimi çalıştırıldığında bir kez yapılır. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		for (int i = 0; i < 10; ++i)
			System.out.println(i);
	}
}
```

for döngüsünün birinci kısmında değişken bildirim yapılabilir. Fakat diğer kısımlarında yapılamaz. Bildirilen değişkene ilk değer verilmesi zorunludur. Örneğin:

```
for (int i = 0; i < 10; ++i)
	<deyim>
```

for döngüsü en çok aşağıdaki biçimde karşımıza çıkar:

```
for (ilkdeğer; karşılaştırma; artırım)
	<deyim>
```

Ancak for döngüsünün bölümleri çok daha farklı şekilde de organize edilebilir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		for (System.out.println("Birinci Kısım"); i < 8; System.out.println("Üçüncü Kısım")) {
			System.out.println("Deyim");
			++i;
		}
	}
}
```

for döngüsünün birinci kısmındaki ifade yazılmayabilir. Bu ifadeyi yukarıya alırsak değişen hiçbir şey olmaz. Örneğin:

```
package csd;

class App {
```

```
	public static void main(String[] args)
	{
		int i = 0;

		for (; i < 10; i++)
			System.out.println(i);
	}
}
```

for döngü deyiminin üçüncü kısmındaki ifade de yazılmayabilir. Bu durumda üçüncü ifade ile yapılması gereken işlem döngü deyiminin sonunda yapılabilir. Örneğin:

```
package csd;

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

		for (int i = 0; i < 10; ) {
			System.out.println(i);
			++i;
		}
	}
}
```

Birinci ve üçüncü kısmı boş olan for döngüleri tamamen while döngüleriyle eşdeğerdir:

```
while (ifade)
	<deyim>
```

ile

```
for (; ifade; )
	<deyim>
```

tamamen eşdeğerdir. Yani biz while döngüleri for gibi, for döngülerini while gibi kullanabilir. Örneğin:

```
for (ifade1; ifade2; ifade3)
	<deyim>
```

ile

```
ifade1;
while (ifade2) {
	<deyim>
	ifade3;
}
```

döngüleri eşdeğerdir.

for döngüsünün ikinci kısmı da hiç yazılmayabilir. Eğer ikinci kısım yazılmazsa koşulun sürekli sağlandığı kabul edilir. for döngüsünde ikinci kısmın yazılmaması adeta burada true değerinin olması gbi bir işlem görmektedir. Yani buraya hiç birşey yazılmaması ile true yazılması tamamen aynı anlamdadır. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		for (int i = 0; ; ++i)
```

```
                        System.out.println(i);
        }
}
```

Burada koşul sürekli sağlandığından sonsuz döngü olur.

for döngüsünde üç kısım da boş bırakılabilir. Ancak iki noktalı virgül bulunmak zorundadır:

```
for (;;)
        <deyim>
```

Burada ikinci kısım boş bırakıldığı için sonsuz döngü olmaktadır. Programcı sonsuz döngü oluşturmak isterse okunabilirlik/algılanabilirlik açısından her üç bölümün de boş bırakıldığı for döngüsünü kullanmalıdır:

```
for (;;) {
        //...
}
```

Bazen for döngüleri yanlışlıkla boş deyim ile kapatılabilmektedir:

```
package csd;

class App {
        public static void main(String[] args)
        {
                for (int i = 0; ; ++i) ; //Dikkat boş deyim
                        System.out.println(i);
        }
}
```

Bu durumda herhangi bir hata oluşmaz. Fakat şüphesiz program hatalı çalışır. Bazı durumlarda döngü deyimlerinde programcının algoritması gereği bilerek boş deyim koyması gerekebilir. Bu durumda programcı okunabilirlik açısından noktalı virgülü bir alt satıra koymalıdır:

```
for (ifade1; ifade2; ifade3)
        ;
```

for döngü deyimi ile belirli bir sayıda dönen örnek döngü kalıpları aşağıdaki biçimlerde oluşturulabilir:

1. n bir tamsayı türünden değişken olmak üzere n-kez dönen döngü şu şekilde oluşturulabilir:

```
for (int i = 0; i < n; ++i)
        <deyim>
```

Örneğin:

```
package csd;

class App {
        public static void main(String[] args)
        {
                int n = 5;
```

```
		for (int i = 0; i < n; ++i)
			System.out.println("CSD");
	}
}
```

2. n bir tamsayı türünden değişken olmak üzere n-kez dönen döngü şu şekilde oluşturulabilir:

for (int i = n - 1; i >= 0; --i)
	<deyim>

Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		int n = 5;

		for (int i = n - 1; i >= 0; --i)
			System.out.println("CSD");
	}
}
```

3. n bir tamsayı türünden değişken olmak üzere n-kez dönen döngü şu şekilde oluşturulabilir:

for (int i = 1; i <= n; ++i)
	<deyim>

Örneğin:

```
package csd;

public class App {

	public static void main(String[] args)
	{
		int n = 5;

		for (int i = 1; i <= n; ++i)
			System.out.println("CSD");
	}
}
```

4. n bir tamsayı türünden değişken olmak üzere n-kez dönen döngü şu şekilde oluşturulabilir:

for (int i = n; i >= 1; --i)
	<deyim>

Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		int n = 5;
```

```
		for (int i = n; i >= 1; --i)
			System.out.println("CSD");
	}
}
```

Anahtar Notlar: Dizilerin ilk elemanlarının indeks numarası 0(sıfır) olduğundan yukarıdaki 1. ve 2. n-kez dönen döngü kalıpları ile bir diziler dolaşılabilmektedir. Diziler (arrays) konusu ileride ele alınacaktır.

for döngüsünün birinci kısmında bildirilen değişkenler döngünün dışında kullanılamazlar. Yalnızca döngünün içerisinde kullanılabilirler. Standartlara göre:

for (bildirim; ifade2; ifade3)
	<deyim>
işleminin tamamen eşdeğeri şöyledir:

{
	bildirim;
	for (; ifade2; ifade3)
		<deyim>
}

Buna göre ayrı döngülerde aynı isimli değişken bildirimi yapılabilir. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		for (int i = 0; i < 10; i++)
			System.out.println(i);

		System.out.println("*****");

		for (int i = 0; i < 10; i++)
			System.out.println(i * i);
	}
}
```

Bir değişken hem döngü içerisinde hem de döngü dışında kullanılmak isteniyorsa bildirim işlemi for döngüsünden önce yapılmalıdır. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		int i;

		for (i = 0; i < 10; i++)
			System.out.println(i);

		System.out.printf("Dongu sonrası i:%d\n", i);
	}
}
```

Burada i değişkeni döngüden önce bildirilmiştir. Döngüden sonra da kullanılabilir. Şüphesiz yukarıdaki örnekte i değişkenine ilk değer verilip döngünün birinci kısmı boş bırakılabilirdi. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		int i = 0;

		for (; i < 10; i++)
			System.out.println(i);

		System.out.printf("Dongu sonrası i:%d\n", i);

	}
}
```

for döngü deyiminin birinci ve üçüncü bölümlerinde virgül atomu kullanılabiir. Birinci bölümde birden fazla bildirim yapılacaksa bildirilecek değişkenler aynı türden olmalıdır. Örneğin:

```
package csd;

class App {
	public static void main(String[] args)
	{
		for (int i = 10, k = 1; i > 1; k++, --i)
			System.out.printf("(%d, %d)\n", i, k);
	}
}
```

Anahtar Notlar: C/C++programlama dillerinde virgül operatörü bulunmaktadır. Java' da virgül bir operatör değildir. for döngüsü için birinci ve üçüncü bölümlerde kullanılabilen bir ayraçtır.

Sınıf Çalışması: Parametresi ile aldığı int türden bir sayının asal olup olmadığını test eden isPrime metodu ve test kodunu yazınız.

Çözüm: (Yavaş versiyon)

```
package csd;

class App {
	public static boolean isPrime(int val)
	{
		if (val <= 1)
			return false;

		int halfVal = val / 2;

		for (int i = 2; i <= halfVal; ++i)
			if (val % i == 0)
				return false;

		return true;
	}

	public static void main(String[] args)
	{
		for (int i = -10; i <= 50; ++i)
			if (isPrime(i))
				System.out.println(i);
	}
}
```

2. Çözüm: (Hızlı versiyon)

package csd;

```
class App {
	public static boolean isPrime(int val)
	{
		if (val <= 1)
			return false;

		if (val % 2 == 0)
			return val == 2;

		if (val % 3 == 0)
			return val == 3;

		if (val % 5 == 0)
			return val == 5;

		if (val % 7 == 0)
			return val == 7;

		if (val % 11 == 0)
			return val == 11;

		for (int i = 13; i * i <= val; i += 2)
			if (val % i == 0)
				return false;

		return true;
	}
	public static void main(String [] args)
	{
		for (int i = -10; i <= 50; ++i)
			if (isPrime(i))
				System.out.println(i);
	}
}
```

Burada matematikteki "Bir sayının asal olması için karakökünden daha küçük asal sayıların hiçbirisine tam olarak bölünmemesi gerekir" teoremi kullanılmıştır.

Sınıf Çalışması: Klavyeden sıfır girilene kadar alınan sayılardan asal olanlarının toplamını, pozitif ve negatif olanların ayrı ayrı sayısını bulan programı yazınız.

Çözüm:

```
package csd;

class App {
	public static boolean isPrime(int val)
	{
		if (val <= 1)
			return false;

		if (val % 2 == 0)
			return val == 2;

		if (val % 3 == 0)
			return val == 3;

		if (val % 5 == 0)
```

```
                return val == 5;

        if (val % 7 == 0)
                return val == 7;

        if (val % 11 == 0)
                return val == 11;

        for (int i = 13; i * i <= val; i += 2)
                if (val % i == 0)
                        return false;

        return true;
    }

    public static void main(String[] args)
    {
        java.util.Scanner kb = new java.util.Scanner(System.in);

        int primeSum = 0, negCount = 0, posCount = 0;

        System.out.println("Sayıları girmeye başlayınız");

        int val;

        while ((val = Integer.parseInt(kb.nextLine())) != 0) {
                if (val > 0) {
                        posCount++;
                        if (isPrime(val))
                                primeSum += val;
                }
                else
                        negCount++;
        }

        System.out.printf("Asal Sayıların Toplamı:%d\n", primeSum);
        System.out.printf("%d tane pozitif sayı girildi\n", posCount);
        System.out.printf("%d tane negatif sayı girildi\n", negCount);

        kb.close();
    }
}
```