Análise e Desenvolvimento de Sistemas – ADS
Programação Orientada a Objetos – POO

Números Aleatórios em Java

A Classe java.util.Random

Assuntos: Métodos; Números Aleatórios.

Prof. Cristóvão Cunha

1. Introdução

Os números aleatórios são utilizados de diversas formas em programas de computador. Eles são importantes no desenvolvimento de jogos, na área de segurança de informações (ex: para gerar senhas ou textos de campos captcha) e em programas de mineração de dados e análise estatística, apenas para citar alguns exemplos.

A linguagem Java disponibiliza funcionalidades para a geração de números aleatórios em uma classe denominada "Random" do pacote "java.util".

2. A Classe Random na Prática

As instâncias da classe "Random" são objetos geradores de números aleatórios, que produzem estes números em resposta a solicitações. A Listagem 1 apresenta um exemplo básico de utilização da classe. A explicação do programa se encontra logo após a especificação do código.

Listagem 1: Exemplo básico de utilização da classe Random

```
import java.util.Random;

public class Random1 {

	public static void main(String[] args) {

		//instância um objeto da classe Random usando o construtor padrão
		Random gerador = new Random();

		//imprime sequência de 10 números inteiros aleatórios
		for (int i = 0; i < 10; i++) {
			System.out.println(gerador.nextInt());
		}
	}
}
```

Um resultado possível para a execução do programa é mostrado na Figura 1.

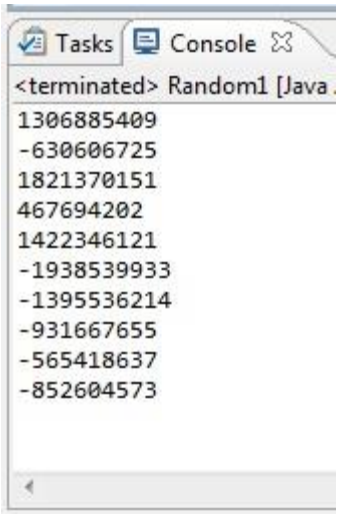

Figura 1: Execução do programa da Listagem 1

Agora vamos à explicação do programa. A classe "Random" está definida no pacote "java.util", portanto qualquer programa Java que faça uso da mesma precisará fazer a importação com "import java.util.Random". No programa da Listagem 1, o gerador de número aleatórios foi instanciado através do uso do **construtor padrão**: *Random gerador = new Random()*.

Esta é a forma mais simples de inicializar um objeto da classe Random. De acordo com a documentação da classe, quando se utiliza o construtor padrão, o valor de **semente** utilizado em cada chamada será "muito provavelmente" diferente do valor de semente escolhido em qualquer invocação posterior desse construtor. A grosso modo, a semente pode ser definida como uma espécie de "ponto de partida" para a geração de uma sequência de números aleatórios (em exemplos posteriores mostraremos como é possível especificar uma semente). O programa contém um loop **for** que gera 10 números aleatórios através de 10 chamadas ao método "**nextInt()**". No exemplo, o método é chamado sem nenhum parâmetro, retornando a cada iteração, um inteiro diferente da sequência de aleatórios. Serão retornados números positivos ou negativos dentre toda a faixa de valores inteiros do Java.

O resumo é o seguinte: todas as vezes que você executar o programa da Listagem 1, obterá uma sequência diferente de 10 números inteiros negativos, zero ou positivos. O menor e o maior número que podem ser gerados são, respectivamente, iguais ao menor e maior inteiro possível na plataforma em que o programa estiver sendo executado.

No entanto, para muitas aplicações práticas você precisará gerar inteiros entre 0 e algum limite superior (ex: 10, 50, 100, etc). Para fazer isto, basta fazer uma chamada ao método "**nextInt()**" passando um parâmetro "**n**". Neste caso, o inteiro a ser retornado estará contido na faixa de 0 a n-1. A Listagem 2 mostra um exemplo de programa que retorna uma sequência de 10 números que sempre possuirão valor entre 0 e 25.

Listagem 2: Gerando sequência de números aleatórios inteiros com valores entre 0 e 25

```
import java.util.Random;

public class Random2 {

	public static void main(String[] args) {

		//instância um objeto da classe Random usando o construtor básico
		Random gerador = new Random();

		//imprime sequência de 10 números inteiros aleatórios entre 0 e 25
		for (int i = 0; i < 10; i++) {
			System.out.println(gerador.nextInt(26));
		}
	}
}
```

Um resultado possível para a execução deste exemplo é mostrado na Figura 2.

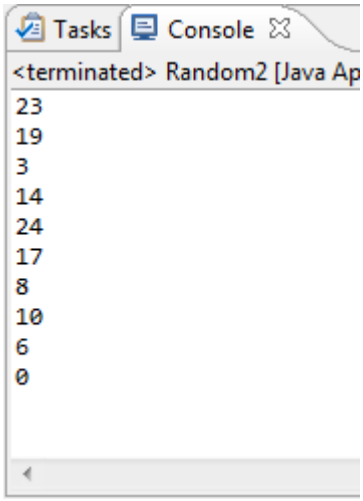

Figura 2: Execução do programa da Listagem 2

Nos exemplos das Listagens 1 e 2, a sequência de números retornada será sempre diferente a cada execução do programa, pois estamos utilizando o construtor default (sem parâmetros) da classe "**Random**". As sequências são diferentes porque quando se usa o construtor default, o Java escolhe "por conta própria" uma semente diferente a cada execução.

Entretanto, também é possível gerar sequências fixas, bastando, para isso, fornecer a sua própria semente. Isto é feito no exemplo da Listagem 3, onde a semente especificada é "19700621".

Listagem 3: Especificando a semente

```
import java.util.Random;

public class Random3 {

	public static void main(String[] args) {

		//instância um objeto da classe Random especificando a semente
		Random gerador = new Random(19700621);

		//imprime sequência de 10 números inteiros aleatórios entre 0 e 25
		for (int i = 0; i < 10; i++) {
			System.out.println(gerador.nextInt(26));
		}
	}
}
```

Neste caso, o programa sempre gerará os mesmos 10 números para a sequência a cada execução (Figura 3).

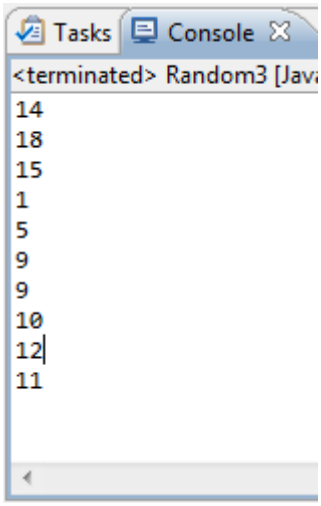

Figura 3: Execução do programa da Listagem 3

A classe "Random" também fornece métodos para a geração de **números reais**: nextDouble() e nextFloat(). Estes métodos **não** aceitam parâmetros, sempre gerando números entre 0 e 1. A Listagem 4 apresenta um exemplo de utilização. No exemplo, a semente não é especificada, porém isso poderia ser feito sem problemas.

Listagem 4: Números reais

```
import java.util.Random;

public class RandomReal {

	public static void main(String[] args) {

		Random r = new Random();

		System.out.println(r.nextDouble());
		System.out.println(r.nextFloat());
	}
}
```

Um exemplo de resultado possível é apresentado na Figura 4.

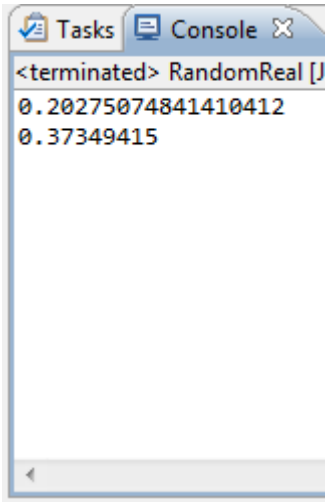

Figura 4: Execução do programa da Listagem 4

A seguir, algumas dicas que poderão ser úteis em situações práticas.

Dica 1: suponha que você precise gerar um número aleatório entre 1 e 6 (ex: para simular um jogo de dados). Se você simplesmente usar o método nextInt(6), obterá números entre 0 e 5, algo que você não deseja. Resolver a situação é simples, bastando aplicar o seguinte recurso:

```
int numDado = gerador.nextInt(6) + 1
```

Dica 2: Imagine que você precise gerar um número real qualquer entre 0 e 90, por exemplo. Vimos que os métodos "nextDouble()" e "nextFloat()" retornam sempre números entre 0 e 1. Então como podemos resolver a situação? A resposta é simples, basta multiplicar o número gerado por 90:

```
double numReal = gerador.nextDouble() * 90
```

Dica 3: Essa é para o pessoal que trabalha com Estatística. Em todos os exemplos apresentados, os números reais e inteiros foram selecionados considerando uma **distribuição uniforme** sobre uma faixa de valores. No entanto, também é possível gerar números aleatórios a partir de uma distribuição gaussiana, com o uso do método "nextGaussian()". Este método não possui parâmetros e retorna um número aleatório considerando uma distribuição normal com média 0 e desvio padrão 1. Para estes valores de média e desvio padrão, cerca de 70% dos números gerados estarão na faixa de -1 a 1 e cerca de 95% entre -2 e 2:

```
double numReal = gerador.nextGaussian()
```

Dica 4: Além de números inteiros e reais, a classe "Random" também pode gerar booleanos:

```
boolean resultado = gerador.nextBoolean()
```

3. Qual a diferença entre o Math.random() e a java.util.Random?

A primeira diferença é que Math.random() é um método estático da classe `Math`, enquanto java.util.Random é uma classe.

Neste aspecto a vantagem de **Math.random** sobre **java.util.Random** é que não é necessário criar um objecto.

Math.random() retorna um `double` de 0,0 até, mas não incluso 1,0. Para conseguir outra faixa de valores é necessário recorrer a operações como a multiplicação e/ou soma. É necessário recorrer ao `cast` para convertê-lo para um número inteiro.

A classe **java.util.Random** fornece formas mais flexíveis para gerar números aleatórios distribuídos uniformemente, proporcionando fácil geração de outros tipos além do `double`.

Um aspecto que pode ser interessante (em caso de testes) é que se duas instâncias de **java.util.Random** forem criados com a mesma semente (`seed`), e a mesma sequência de chamadas de métodos for feita para cada uma, elas vão gerar e retornar sequências idênticas de números.

Em resumo:

A classe é mais completa e flexível, permite determinar o tipo de dado que se deseja e é mais eficiente. Ela pode, por exemplo, repetir os mesmos números se usada com a mesma semente.

O método estático contido em `Math` é mais simples e resolve sem muitas preocupações, o método cuida de vários detalhes para você, consequentemente você não pode configurar como quer usar. Só trabalha com valores `double`. Ele é prático quando precisa do básico.

4. Conclusão

Este texto é uma breve introdução à classe Random. Através dela, é possível gerar diversos tipos distintos de números aleatórios e nem todos foram aqui apresentados. Para obter informações adicionais, consulte http://docs.oracle.com/javase/7/docs/api/java/util/Random.html.