2 – Utilizando Objetos

Nesse capitulo iremos aprender a manipular objetos que pertencem as classes predefinidas do Java. Esse conhecimento irá prepará-lo para aprender a implementar suas próprias classes.

2.1 - Tipos e Variáveis

Em Java, cada valor é de um tipo.

Em Java, cada valor *é* de um **tipo**. Por exemplo, "Hello, World" é do **tipo** String, o objeto System.out é do tipo PrintStream e o número 13 é do tipo int (uma abreviação para "integer", ou inteiro). O **tipo** informa o que você pode fazer com os valores. Você pode chamar println em qualquer **objeto** do **tipo** PrintStream. Pode também calcular a soma ou o produto de dois inteiros quaisquer.

Usa-se variáveis para armazenar valores que se deseja utilizar em um momento posterior.

É muito comum querermos armazenar os valores para utilizá-los posteriormente. Para lembrar-se de um **objeto**, você precisa armazená-lo em uma ***variável***. Uma **variável** é um local de armazenamento na memória do computador que possui um ***tipo***, um ***nome*** e um ***conteúdo***. Por exemplo, aqui declaramos três **variáveis**:

```
String msg = "Hello, World!";

PrintStream printer = System.out;

int idade = 28;
```

A primeira **variável** chama-se msg. Ela pode ser utilizada para armazenar valores do **tipo** String e é configurada com o valor "Hello, World!". A segunda **variável** armazena um valor do **tipo** PrintStream e a terceira armazena um inteiro.
Variáveis podem ser utilizadas no lugar dos **objetos** que elas armazenam:

```
printer.println(msg) ;// O mesmo que System.out.println("Hello, World!")

printer.println(idade) ; // O mesmo que System.out.println(28)
```

Definição: variável

```
nomeDoTipo nomeDaVariável = valor ;
                ou
nomeDoTipo nomeDaVariável ;
```

Exemplo

```
String msg = "Hello, World" ;
```

Objetivo

Definir uma nova variável de um tipo particular e, opcionalmente, fornecer um valor inicial.

Ao declarar suas próprias **variáveis**, você precisa tomar duas decisões.

- Qual **tipo** você deve utilizar para a **variável**?
- Qual **nome** você deve atribuir à **variável**?

O **tipo** depende do uso final. Se precisar armazenar uma string, utilize o **tipo** String para sua **variável**. É um erro armazenar um valor cuja classe não corresponde ao **tipo** da **variável**. Por exemplo, *o seguinte é um erro*:

```
String msg = 13; // ERRO: Tipos incompatíveis
```

Você não pode utilizar uma **variável** String para armazenar um inteiro. O compilador verifica não-correspondências de tipo para protegê-lo contra erros.

Identificadores para variáveis, métodos e classes são compostos de letras, dígitos e caracteres de sublinhado.

Ao decidir sobre um nome para uma **variável**, você deve fazer uma escolha que descreve o propósito da **variável**. Por exemplo, o nome da **variável** msg é uma escolha melhor do que o nome xYz.

Um ***identificador*** é o nome de uma **variável**, **método** ou **classe**. Java impõe as seguintes regras para identificadores:

- Identificadores podem ser compostos de *letras*, *dígitos*, *caracteres de sublinhado (_) e sinal de cifrão ($)*. Eles, porém, não podem iniciar com um dígito. Por exemplo, msg1 é válido, mas 1msg não.
- Você não pode utilizar outros símbolos como ? ou %. Por exemplo, hello! não é um identificador válido.
- Não são permitidos espaços em identificadores. Portanto, nome da mae não é válido.
- Além disso, você não pode utilizar *palavras reservadas*, como public como nomes; essas palavras são reservadas exclusivamente para seus significados especiais em Java.
- Identificadores também fazem distinção entre letras maiúsculas e minúsculas; isto é, idade e Idade são diferentes.

Essas são regras rígidas da linguagem Java. Se violar uma delas, compilador informará um erro. Além disso, há algumas **convenções** que você deve obedecer para que seus programas possam ser lidos facilmente por outros programadores.

- Nomes de **variáveis** e **métodos** devem iniciar com *letra minúscula*. É válido utilizar uma letra **maiúscula** ocasionalmente, como nomeDaMae. Essa combinação de *letras minúsculas* e *maiúsculas* às vezes é chamada de "*notação camelo*" porque as letras *maiúsculas* se destacam como a corcova de um camelo.
- Nomes de **classes** devem iniciar com *letra maiúscula*. Por exemplo, HelloWorld seria um nome apropriado para uma **classe**, *mas não para uma variável*.

Se violar essas **convenções**, o compilador não reclamará, mas você irá confundir outros programadores que lêem seu código.

2.2 - Operador de Atribuição

Utilize o operador de atribuição "=" para alterar o valor de uma variável.

Você pode alterar o valor de uma **variável** existente com o **operador** de **atribuição** "=". Por exemplo, considere a definição da **variável** a seguir

`int idade = 48;` Se quiser alterar o valor dessa **variável**, simplesmente **atribua** o novo valor:

```
idade = 18;
```

A **atribuição** substitui o valor original da **variável**. Na linguagem de programação Java, o **operador** = denota uma ação, substituir o valor de uma **variável**. Esse uso difere do uso tradicional do símbolo =, como um **operador** de igualdade.

É um erro utilizar uma **variável** à qual nunca foi **atribuído** um valor. Por exemplo, a seqüencia de instruções representa um erro comum.

```
int idade;
System.out.println(idade);      // ERRO - variável não-inicializada
```

Todas as variáveis devem ser inicializada antes de você utiliza-la.

O compilador reclama sobre uma "variável não-inicializada" quando você usa uma **variável** à qual nunca foi **atribuído** valor algum.

O correto é **atribuir** um valor à **variável** *antes* de utilizá-la:

```
int idade;
idade = 13;
System.out.println(idade);
```

Ou, melhor ainda, **inicialize** a **variável** ao defini-la.

```
int idade = 13;
System.out.println(idade) ;  // OK
```

Definição: Atribuição

```
nomeDaVariável = valor ;
```

Exemplo

```
idade = 19 ;
```

Objetivo

Definir um novo valor a uma variável previamente definida.

2.3 - Objetos, classes e métodos

Objetos são entidades no seu programa que você manipula invocando métodos.

Um **objeto** é uma entidade que você pode manipular no seu programa. Normalmente, você não sabe como o **objeto** é organizado internamente, mas ele tem um **comportamento** bem-definido e é isso o que nos importa quando o utilizamos.

Você manipula um **objeto** chamando um ou mais dos seus **métodos**. Um **método** consiste em uma seqüencia de instruções que acessam os dados internos. Quando você chama o **método**, não há como saber exatamente quais são essas **instruções**, mas você sabe o *propósito* do **método**.

Um método é uma sequencia de instruções que acessam os dados de um objeto.

Por exemplo, vimos que System.out refere-se a um **objeto**. Você manipula chamando o método println. Quando o método println é chamado, algumas atividades ocorrem dentro do **objeto** e o efeito final é que o *texto aparece na janela da console*. Você não sabe como isso acontece, mas isso é válido. O importante é que o **método** execute aquilo que você solicitou.

Figura - Representação do objeto System.out

A Figura mostra uma representação do objeto System.out. Os dados internos são simbolizados por uma seqüencia de zeros e uns. Pense em cada **método** (simbolizado pelas engrenagens) como uma parte de maquinaria que executa uma determinada tarefa.

Na primeira aula vimos dois objetos:

1. System.out
2. “Hello, World!”

Esses **objetos** pertencem a diferentes **classes**. O objeto System.out pertence à classe PrintStream. O objeto “Hello, World!” pertence à classe String.

Uma **classe** especifica os **métodos** que você pode aplicar aos **objetos** dela. Você pode utilizar o **método** println com qualquer **objeto** pertencente à **classe** PrintStream. System.out é um desses **objetos**. É possível obter outros **objetos** da **classe** PrintStream. Por exemplo, você pode construir um **objeto** PrintStream com o objetivo de enviar saída para um arquivo. A **classe** String fornece **métodos** que você pode aplicar a **objetos** String. Um deles é o **método** length. O **método** length conta o número de caracteres em uma string. Você pode aplicar esse **método** a qualquer **objeto** do tipo String. Por exemplo, a seqüencia de instruções

```
String msg = “Hello, World!”;
int quantosCaracteres = msg.length();
```

inicializa quantosCaracteres com o número de caracteres do **objeto** String “Hello, World!”. Depois de as instruções no método length serem executadas, quantosCaracteres é configurado como 13. (As aspas na são parte da string e o **método** length não as conta).

Figura - Representação de um objeto do tipo String.

O **método** length - diferentemente do **método** println - não requer entrada dentro dos parênteses. Entretanto, o método length fornece uma saída, a saber, a contagem de caracteres.

2.4 - Parâmetros de métodos e valores de retorno

Agora, você verá mais detalhadamente como fornecer entradas a um **método** e obter saídas do **método**. Vejamos outro **método** da **classe** String. Quando você aplica o **método** toUpperCase a um **objeto** String, esse **método** cria outro **objeto** String que contém os caracteres da string original, com as letras minúsculas convertidas em maiúsculas. Por exemplo, a seqüencia de instruções

```
String cidade = "São Paulo";
String grandeCidade = cidade.toUpperCase();
```

configura grandeCidade como o **objeto** String "SÃO PAULO".

Ao aplicar um **método** a um **objeto**, você deve certificar-se de que esse **método** esteja definido na **classe** apropriada. Por exemplo, é um erro chamar

```
System.out.length () ; // Essa chamada de método é um erro.
```

A classe PrintStream (à qual System.out pertence) não possui um **método** length.

Alguns **métodos** requerem entradas que fornecem detalhes sobre o trabalho que precisam fazer. Por exemplo, o método println tem uma entrada: a string que deve ser impressa. Cientistas da computação utilizam o termo técnico **parâmetro** para entradas de **método**. Dizemos que a string msg é um **parâmetro** da chamada de **método** System.out.println(msg).

Tecnicamente falando, o **parâmetro** msg é um **parâmetro explícito** do método println. O **objeto** em que você invoca o **método** também é considerado um **parâmetro** da chamada de **método**, e é denominado **parâmetro implícito**. Por exemplo, System.out é o **parâmetro implícito** da seguinte chamada de **método**:

```
System.out.println(msg);
```

Alguns **métodos** requerem múltiplos **parâmetros explícitos**, outros não requerem absolutamente nenhum. Um exemplo do último é o **método** length da **classe** String. Todas as informações que o **método** length requer para fazer o trabalho - a saber, a seqüencia de caracteres da string - estão armazenadas no próprio **objeto parâmetro implícito**.

O **método** length difere do **método** println de uma outra maneira: ele tem uma saída. Dizemos que o **método retorna um valor**, a saber, o número de caracteres na string. Você pode armazenar o valor de **retorno** em uma **variável**:

```
int n = msg.length();
```

Você também pode utilizar o valor de retorno como um parâmetro de outro método:

```
System.out.println(msg.length());
```

A chamada de **método** msg.length() retorna um valor - o inteiro 13. O valor de **retorno** torna-se um **parâmetro** do **método** println.

Nem todos os **métodos** retornam valores. Um exemplo é o **método** println. O **método** println interage com o sistema operacional, fazendo com que os caracteres apareçam em uma janela. Mas ele não retorna um valor ao código que o chama.

Vamos analisar uma chamada de **método** mais complexa. Aqui, chamaremos o **método** replace da **classe** String. O **método** replace executa uma operação de pesquisa e substituição, semelhante àquela de um

processador de texto. Por exemplo, a chamada:

```
cidade. replace("Paulo", "Sebastião")
```

constrói uma nova string que é obtida substituindo todas as ocorrências de "Paulo" em "São Paulo" por "Sebastião". O **método** retorna o **objeto** String "São Sebastião" (que tanto pode ser salvo em uma **variável** como passado para outro **método**).

Quando um **método** é definido em uma **classe**, essa definição especifica os tipos dos **parâmetros** explícitos e o valor de **retorno**. Por exemplo, a **classe** String define o **método** length como:

```
public int length()
```

Isto é, não há **parâmetro explícito** e o valor de **retorno** é do **tipo** int. (Por enquanto, todos os **métodos** que consideramos serão **métodos publicos**) . O **tipo** do **parâmetro implícito** é a **classe** que define o **método** - String no nosso caso. Ele não é mencionado na definição de **método** - daí o termo "implícito".

O **método** replace é definido como:

```
public String replace(String antigoValor, String novoValor)
```

Para chamar o **método** replace, você fornece dois **parâmetros explícitos**, antigoValor e novoValor, que são do tipo String. O valor retornado é uma outra string.

Quando um **método** não retorna valor algum, o **tipo** de **retorno** é declarado com a palavra reservada **void**. Por exemplo, a **classe** PrintStream define o **método** println como:

```
public void println(String output)
```

Ocasionalmente, uma **classe** define dois **métodos** com o mesmo nome e diferentes **tipos** de **parâmetros explícitos**. Por exemplo, a **classe** PrintStream define um segundo **método**, também chamado println, como:

```
public void println(int output)
```

Um nome de método é sobrecarregado se uma classe tiver mais de um método com o mesmo nome (mas tipos diferentes de parâmetros).

Esse **método** é utilizado para imprimir um valor inteiro. Dizemos que o nome println é **sobrecarregado** porque referencia mais de um **método**.

2.5 - Tipos Numéricos

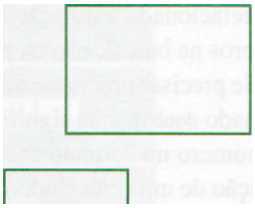
Java possui tipos separados para **inteiros** e números de **ponto flutuante**. **Inteiros** são números sem parte fracionária; números de **ponto flutuante** podem ter partes fracionárias. Por exemplo, 13 é um **inteiro** e 1.3 é um número de **ponto flutuante**.

Há vários outros tipos numéricos em Java que veremos nas próximas aulas. Em Java, os tipos numéricos (int, double) são **tipos primitivos**, não **classes**.

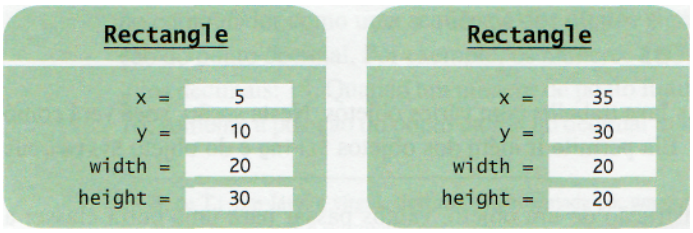
Você também pode combinar números a operadores, por exemplo, + e -, como em 10 + n ou n - 1. Para multiplicar dois números, utilize o operador *. Como na matemática, o operador * tem precedência sobre o operador +. Isto é, x + y * 2 significa a soma de x com y * 2. Se desejar multiplicar a soma de x e y por 2, use parênteses: (x + y) * 2.

2.6 – Construindo Objetos

A maioria dos programas Java trabalha com vários **objetos**. Nesta seção, você verá como construir novos **objetos**. Ela permite ir além dos objetos String e do objeto System.out predefinido.
Para entender a construção de um **objeto**, vamos passar para uma outra **classe**: a classe Rectangle na biblioteca de **classes** Java. Os objetos do tipo Rectangle descrevem formas retangulares (Figura abaixo).



Observe que um **objeto** Rectangle não é uma forma retangular - ele é um **objeto** que contém um conjunto de números. Os números *descrevem o* retângulo (veja Figura abaixo). Cada retângulo é descrito pelas coordenadas *x* e *y* do seu canto superior esquerdo, sua largura e sua altura.



É muito importante que você entenda essa distinção. No computador, um **objeto** Rectangle é um bloco de memória que armazena quatro números, por exemplo, *x = 5, y = 10, largura = 20, altura = 30*. Na imaginação do programador que usa um **objeto** Rectangle, esse **objeto** descreve uma figura geométrica.

Utilize o operador new, seguido por um nome de classe e parâmetros para construir novos objetos.

Para criar um novo retângulo, você precisa especificar os valores de *x, y, largura* e *altura*. Então *invoque o* operador **new**, especificando o nome da **classe** e os **parâmetros** necessários para construir um novo objeto. Por exemplo, você pode criar um novo retângulo com o canto superior esquerdo em (5,10), largura 20 e altura 30 como a seguir:

```
new Rectangle(5, 10, 20, 30);
```

Eis o que acontece em detalhe.

1. O operador **new** cria um **objeto** Rectangle.
2. Usa os **parâmetros** (nesse caso, 5,10, 20 e 30) para **inicializar** os dados do **objeto**.
3. Retorna o **objeto**.

Normalmente, a saída do operador **new** é armazenada em uma **variável**. Por exemplo,

```
Rectangle box = new Rectangle(5, 10, 20, 30);
```

O processo da criação de um novo **objeto** é chamado *construção*. Os quatro valores 5, 10, 20 e 30 são chamados *parâmetros de construção*. Observe que a expressão **new** *não é* uma instrução completa. Você usa o valor de uma expressão **new** da mesma maneira que um valor de retorno de **método**: Atribua-o a uma **variável** ou passe-o para outro **método**.

Algumas **classes** permitem construir **objetos** de várias maneiras. Por exemplo, você também pode obter um **objeto** Rectangle sem fornecer um **parâmetro** de construção (mas você ainda deve fornecer os parênteses):

```
new Rectangle();
```

Essa expressão constrói um retângulo (bastante inútil) com o canto superior esquerdo na origem (0, 0), largura 0 e altura 0.

Definição: Construção de Objetos

```
new NomeDaClasse(parâmetros);
```

Exemplo

```
new Rectangle(5, 10, 20, 30);
new Rectangle();
```

Objetivo

Construir um novo objeto. Inicialize-o com os parâmetros de construção e retorne uma referência ao objeto construído

2.7 – Métodos de acesso e métodos modificadores

Nesta seção, introduziremos uma terminologia útil para os **métodos** de uma **classe**. Um **método** que acessa um **objeto** e retorna algumas informações sobre ele, sem alterá-lo, é chamado de *método de acesso*. Em comparação, um método cujo propósito é modificar o estado de um objeto é chamado de método *modificador.*

Por exemplo, o método length da classe String é um **método** de acesso. Ele retorna informações sobre uma string, a saber, seu comprimento. Mas ele não modifica a string ao contar os caracteres.
A classe Rectangle tem alguns métodos de acesso. Os **métodos** getX, getY, getWidth e getHeight retornam as coordenadas *x* e *y* do canto superior esquerdo e os valores de largura e altura. Por exemplo,

```
double width = box.getWidth();
```

Agora vamos considerar um **método** modificador. Programas que manipulam retângulos frequentemente precisam movimentá-los para, por exemplo, exibir animações. A classe Rectangle tem um **método** para esse propósito, chamado translate (Matemáticos utilizam o termo "translação" para um movimento rígido do plano). Esse **método** move um retângulo por certa distância nas direções *x* e *y*. A chamada de **método**:

```
box.translate(15, 25);
```

move o retângulo por 15 unidades na direção x e 25 unidades na direção y (veja Figura abaixo). Mover um retângulo não muda a largura nem a altura, mas altera o canto superior esquerdo. Após o movimento, o canto superior esquerdo estará em (20, 35). Esse é um método modificador porque altera o objeto parâmetro implícito.

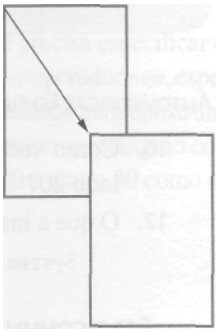


Figura – Utilizando o método translate para mover um retângulo.

2.8 – Implementando um programa de teste

Nesta seção, discutiremos os passos necessários para implementar um programa de teste. O propósito de um programa de teste é verificar se um ou mais **métodos** foram implementados corretamente. Um programa de teste chama **métodos** e verifica se eles retornam os resultados esperados. Escrever programas de teste é uma atividade muito importante. Ao implementar seus próprios **métodos**, você sempre deve fornecer programas para testá-los.

Neste curso, utilizaremos um formato muito simples para programas de teste. Você verá agora um desses programas que testa um **método** na **classe** Rectangle. Esse programa realiza as seguintes ações:

- Fornece uma **classe** testadora.
- Provê um **método main**.
- Dentro do **método main**, constrói um

ou mais **objetos**.

- Aplica **métodos** aos **objetos**.
- Exibe os resultados das chamadas de **método**.
- Exibe os valores que você espera obter.

Você precisa seguir esses passos sempre que escrever um programa para testar suas **classes**. Nosso programa de teste de exemplo testa o comportamento do **método** translate. Eis os principais passos (que foram colocados dentro do **método main** da **classe** TesteRetangulo).

```
Rectangle box = new Rectangle(5, 10, 20, 30);
// Move o retângulo
box.translate(15, 25);
// Imprime informações sobre o retângulo movido
System.out.print("x: ");
System, out.println (box. getX()) ;
System.out.println("Esperado: 20");
```

Imprimimos o valor retornado pelo **método** getX e então imprimimos uma mensagem que descreve o valor que esperamos ver.

Determinar o resultado esperado antecipadamente é uma parte importante do processo de teste.

Esse é um passo muito importante. Pare um pouco para pensar qual é o resultado esperado antes de você executar um programa de teste. Essa consideração prévia do processo irá ajudá-lo a entender como seu programa deve se comportar e pode ajudá-lo a rastrear erros no começo do desenvolvimento de um projeto. No nosso caso, o retângulo foi construído com o canto esquerdo superior em (5, 10). A direção *x* foi movida por 15 *pixels*, portanto esperamos um valor *x* de 5 + 15 = 20 depois dessa movimentação.

Eis um programa completo que testa a movimentação de um retângulo

```
import java.awt.Rectangle;

public class TesteRetangulo
{
	public static void main(String[] args)
		Rectangle box = new Rectangle(5, 10, 20, 30);

		// Imprime informações sobre o retângulo movido
		System.out.print("x: ");
		System.out.println(box.getX());
		System.out.println("Esperado: 20");
		System.out.print("y: ");
		System.out.println(box.getY());
		System.out.println("Esperado: 35");
	}
}
```

Classes Java são agrupadas em pacotes. Utilize a instrução import para usar classes definidas em outros pacotes.

Para esse programa, precisamos seguir outro passo: Precisamos **importar** a **classe** Rectangle a partir de *um* **pacote**. Um **pacote** é uma coleção de **classes** com um propósito relacionado. Todas as classes na biblioteca padrão estão contidas em **pacotes**. A classe Rectangle pertence ao **pacote** java.awt, que contém muitas classes para desenhar janelas e formas gráficas.

Para utilizar a **classe** Rectangle no pacote java. awt, simplesmente posicione a linha a seguir na parte superior do seu programa:

```
import java.awt.Rectangle;
```

Por que não precisa **importar** as **classes** System e String? Porque as **classes** System e String estão no **pacote** java.lang,

e todas as classes nesse pacote são automaticamente importadas, assim você nunca precisará importá-las.

Definição: Importando uma classes a partir de um pacote

```
import nomeDoPacote.nomeDaClasse;
```

Exemplo

```
import java.awt.Rectangle;
```

Objetivo

Importar uma classe a partir de um pacote para uso em um programa

2.9 – A documentação da API

As classes e métodos da biblioteca Java estão listados na *documentação da API*. A API *(application programming interface) é* a interface de programação de aplicativo. Um programador que usa as classes Java para montar um programa de computador (ou *aplicativo) é* um *programador de aplicativo*. Esse é você. Em comparação, os programadores que projetaram e implementaram as classes de biblioteca como PrintStreatn e Rectangle *são programadores de sistema*.
Você pode encontrar a documentação da API na Web (http://java.sun.com/javase/6/docs/api/index.html). Alternativamente, você pode fazer download e instalar a documentação da API em seu próprio computador. A documentação da API documenta todas as classes na biblioteca Java - há milhares delas. A maioria das classes é bastante especializada e somente algumas têm interesse ao programador iniciante.

2.10 – Referências a objetos

Em Java, uma **variável** cujo **tipo** é uma **classe**, na verdade não contém um **objeto**. Ela contém meramente *aposição de um objeto na memória*. O objeto em si é armazenado em uma outra parte. Utilizamos o termo técnico **referência a objeto** significando a posição de um **objeto** na memória. Quando uma **variável** contém a posição de um **objeto** na memória, dizemos que ela **referência** um **objeto**. Por exemplo, depois da instrução

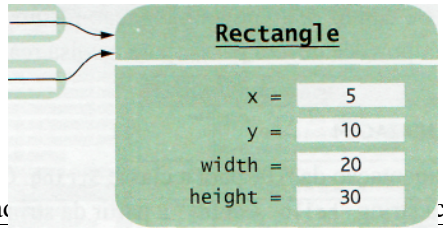
Uma referência descreve a localização de um objeto.

```
Rectangle box = new Rectangle(5, 10, 20, 30);
```

a **variável** box referencia o **objeto** Rectangle que o operador **new** construiu. Tecnicamente falando, o operador **new** retornou uma **referência** ao novo **objeto** e essa referência é armazenada na **variável** box.
É muito importante lembrar-se de que a **variável** box não contém o **objeto**. Ela **referencia** o **objeto**. Você pode fazer duas variáveis de objeto referenciarem o mesmo objeto:

```
Rectangle box2 = box;
```

Agora você pode acessar o mesmo **objeto** Rectangle como box ou como box2 conforme mostrado na Figura abaixo.



Mas **variáveis** numéricas, na verdade [illegible] você define

```
int idade = 19;
```

a **variável** idade contém o número 19, não uma referência ao número.

Variáveis numéricas armazenam números. Variáveis de objeto armazenam referências.

Você pode ver a diferença entre **variáveis** numéricas e **variáveis** de **objeto** ao criar uma cópia de uma **variável**. Ao copiar um valor de **tipo primitivo**, o número original e a cópia do número são valores independentes. Mas quando você copia uma **referência a objeto**, tanto o original como a cópia são **referências** ao mesmo **objeto**.

Há uma razão para essa diferença entre números e objetos. No computador, cada número requer uma pequena quantidade de memória para seu armazenamento. Mas os objetos podem ser muito grandes. É muito mais eficiente manipular somente a sua posição na memória.

Francamente falando, a maioria dos programadores não se preocupa muito com a diferença entre objetos e referências a objeto. Boa parte do tempo, você terá a intuição correta ao pensar no "objeto box" em vez de "a referência ao objeto armazenada em box". tecnicamente mais exata. A diferença entre objetos e referências a objeto só é aparente quando você tem múltiplas variáveis que se referem ao mesmo objeto.

2.11 – Exercícios

1) Qual é o tipo dos valores 19 e "19"?

2) Quais dos seguintes são identificadores válidos?
a) Hello1
b) void
c) 101x
d) Hello, World
e) <helloworld>

3) Defina uma variável para armazenar seu nome. Utilize a notação camelo no nome da variável.

4) 12 = 12 é uma expressão válida na linguagem Java?

5) Como você altera o valor da variável msg para "Oi Java!"?

6) Como você pode calcular o comprimento da string "Rio de Janeiro"?

7) Como você pode imprimir a versão em letras maiúsculas de "Hello, World!"?

8) É válido chamar cidade.println()? Justifique sua resposta.

9) Quais são os parâmetros implícitos, os parâmetros explícitos e os valores de retorno na chamada do método cidade. length () ?

10) Qual é o resultado da chamada cidade.replace("S", "D")?

11) Qual é o resultado da chamada msg.replace("World", "Brazil"). length()?

12) Como o método toUpperCase é definido na classe String?

13) Qual tipo de número você utilizaria para armazenar a área de um círculo?

14) Por que a expressão 13.println() é um erro?

15) Escreva uma expressão para calcular a média dos valores x e y

16) Como você constrói um quadrado com centro (100, 100) e comprimento de lado 20?

17) O que a instrução a seguir imprime?

```
System.out.println (new Rectangle().getWidth());
```

18) O método toUpperCase da classe String é um método de acesso ou um modificador?

19) Que chamada ao método translate é necessária para mover o retângulo box de modo que seu canto superior esquerdo seja a origem (0, 0)?

20) Suponha que box.translate(25, 15) tenha sido chamada em vez de box.translate(15, 25). Quais são as saídas esperadas?

21) Por que o programa TesteRetangulo não precisa imprimir a largura e a altura do retângulo?

22) A classe Random é definida no pacote java.util. O que você precisa fazer para utilizar essa classe no seu programa?

23) Examine a documentação da API para a classe String. Qual método você utilizaria para obter a string "hello, world!" a partir da string "Hello, World!"?

24) Na documentação da API para a classe String, examine a descrição do método trim. Qual é o resultado da aplicação de trim à string “ Hello, Space !”

25) Qual é o efeito da atribuição msg2 = msg?

26) Depois de chamar msg2.toUpperCase(), qual é o conteúdo de msg e msg2