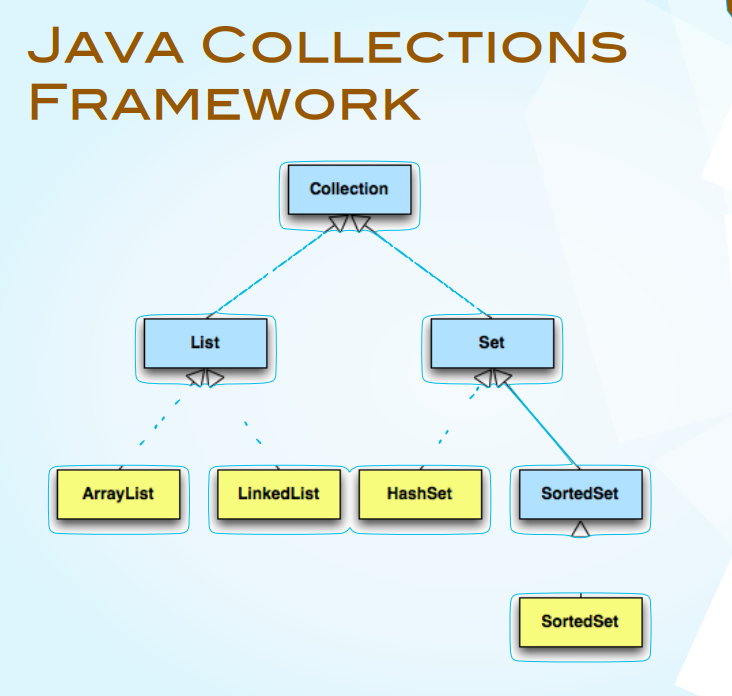
Estrutura de Dados



Genérico:

Uma classe que é definida à custa de um tipo parametrizado é denominada de classe genérica ou classe parametrizada

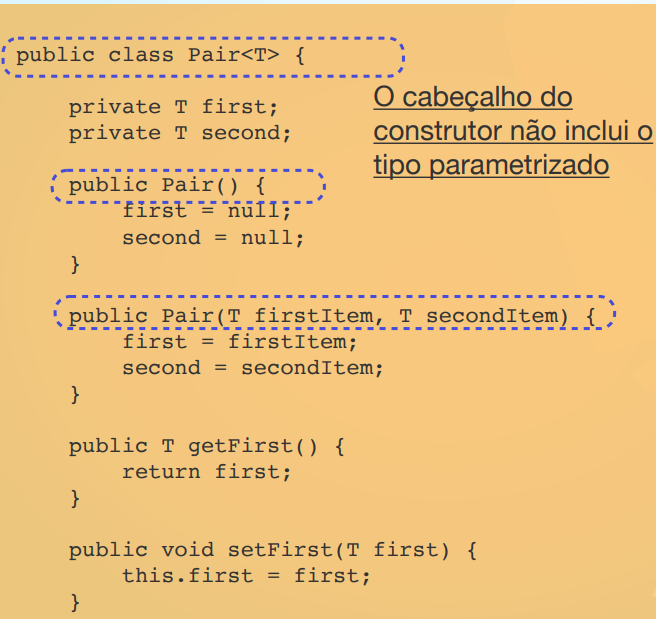
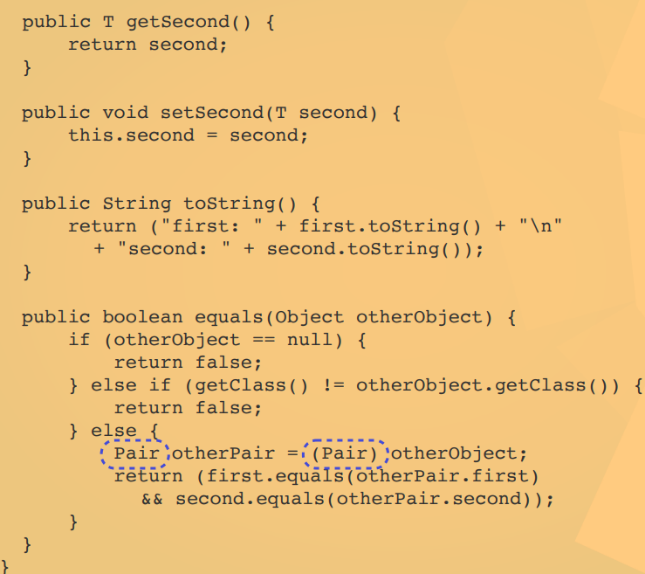
+ O tipo parametrizado é incluído entre "<>" depois do nome da classe no cabeçalho da definição da classe

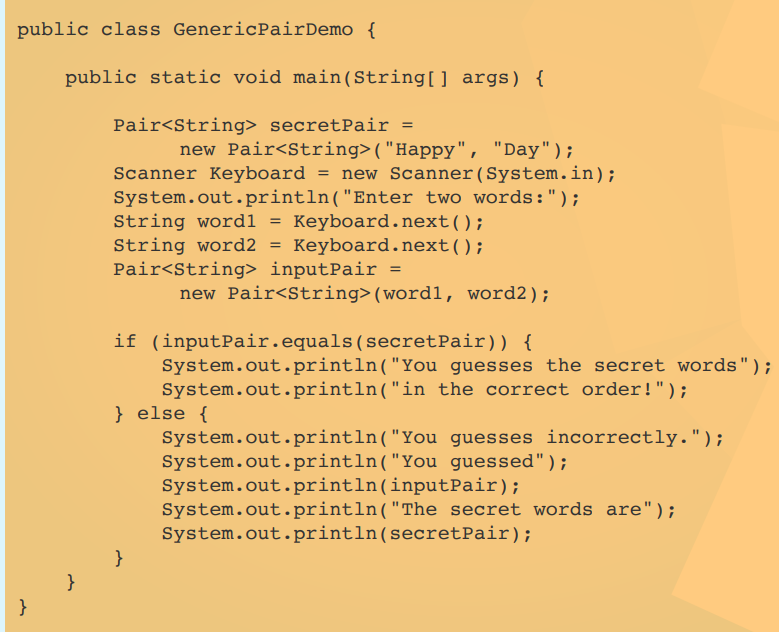
+ Pode ser usado qualquer identificador desde que não seja uma palavra reservada para o tipo, mas por convenção, o parâmetro começa com uma letra maiúscula

+ O tipo parametrizado pode ser usado como outros tipos usados na definição da classe

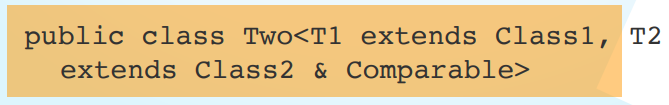
Imagine que você tem uma caixa de brinquedos, e essa caixa pode ser usada para guardar diferentes tipos de brinquedos, como carrinhos, bonecas ou bolas. Chamaríamos essa caixa de "caixa de brinquedos genérica" porque pode conter qualquer tipo de brinquedo, não importa o que seja. É uma maneira legal de armazenar coisas diferentes em um só lugar.

No mundo da programação, "genérico" é como essa caixa de brinquedos. É uma maneira de criar coisas que podem funcionar com diferentes tipos de dados, como números, palavras ou qualquer outra coisa. É como se você tivesse uma ferramenta mágica que pode ser usada para lidar com qualquer tipo de informação.





Limites para tipos parametrizados



A declaração public class Two<T1 extends Class1, T2 extends Class2 & Comparable> em Java de uma forma simples.

Imagine que você está criando uma caixa (classe) que pode conter dois tipos diferentes de coisas (objetos). No entanto, você quer definir regras específicas para os tipos de coisas que podem ser colocadas na caixa.

T1 extends Class1: Isso significa que o primeiro tipo T1 deve ser um subtipo (ou o próprio tipo) da classe Class1. Em outras palavras, apenas coisas que são "tipo Class1" ou seus subtipos podem ser colocadas na primeira parte da caixa.

T2 extends Class2 & Comparable: Isso significa que o segundo tipo T2 deve ser um subtipo (ou o próprio tipo) da classe Class2, mas também deve implementar a interface Comparable. Isso significa que apenas coisas que são "tipo Class2" e que podem ser comparadas entre si podem ser colocadas na segunda parte da caixa.

Essas regras ajudam a garantir que, ao usar essa classe, você só possa colocar objetos que atendam a esses critérios específicos na caixa. Isso pode ser útil para criar classes genéricas que funcionam apenas com tipos de dados que atendem a determinados requisitos, tornando seu código mais seguro e controlado.

Métodos Genéricos:

Quando uma classe genérica é definida o tipo parametrizado pode ser usado nas definições de métodos para a classe genérica

+ Um método genérico pode ainda ser definido com os seus próprio tipos parametrizados sem que sejam os mesmo da classe

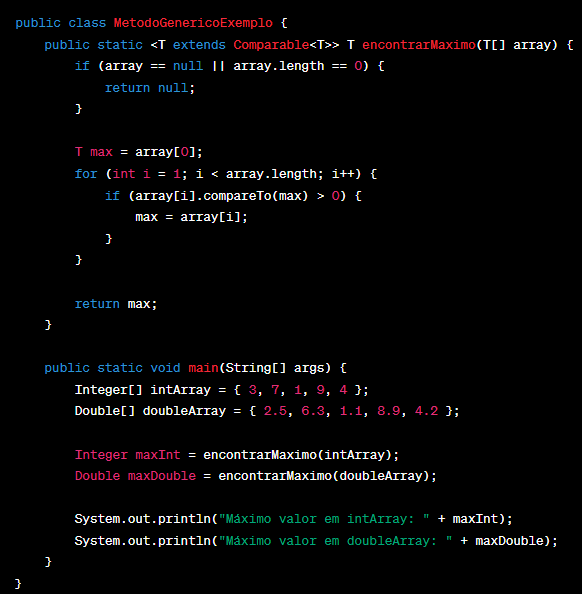
+ Um método genérico pode ser um membro de uma classe normal ou genérica que tenha algum tipo parametrizado

+ O tipo parametrizado de um método genérico é local a esse método e não à classe 34

+ O tipo parametrizado tem de ser colocado (entre "<>") depois de todos os modificadores e antes do tipo de retorno

Imagine que você tem uma máquina mágica que pode fazer diferentes tipos de sucos, como suco de maçã, suco de laranja ou suco de uva. No entanto, você não quer criar uma máquina separada para cada tipo de suco, pois isso seria trabalhoso.

Em vez disso, você cria uma máquina "genérica" que pode fazer qualquer tipo de suco, desde que você lhe diga o que colocar dentro. Por exemplo, você pode dizer: "Faça suco de maçã" ou "Faça suco de laranja" e a máquina faz o suco certo com base no que você pediu.



Herança com Classes Genéricas:

Uma classe genérica pode ser definida como subclasse de uma classe normal ou mesmo de uma outra classe genérica

+ Tal como nas classes normais um objecto do tipo de uma subclasse será também do tipo da superclasse

+ Dadas duas classes: A e B, e uma classe genérica: G não existe nenhum relacionamento entre G e G

**+ É sempre verdade independente do relacionamento entre as classes A e B, por exemplo, se a classe B é uma subclasse da classe A**

Imagine que você tem uma família de robôs, cada um projetado para realizar tarefas específicas. Você tem um robô de limpeza, um robô de cozinha e um robô de jardinagem. Todos esses robôs compartilham algumas características em comum, como a capacidade de se mover e executar ações, mas cada um deles também tem suas próprias habilidades exclusivas.

Em programação, a herança com classes genéricas é como essa família de robôs. Você pode criar uma classe genérica que contém características e funcionalidades comuns a vários tipos de objetos. Em seguida, você pode criar classes especializadas que herdam (ou seja, obtêm) essas características da classe genérica e adicionam suas próprias funcionalidades específicas.

Por exemplo, você pode ter uma classe genérica chamada Robo que contém métodos para mover e executar ações básicas. Em seguida, você pode criar classes especializadas, como RoboDeCozinha e RoboDeJardinagem, que herdam as funcionalidades do Robo e também têm métodos adicionais para suas tarefas específicas.



Wildcards:

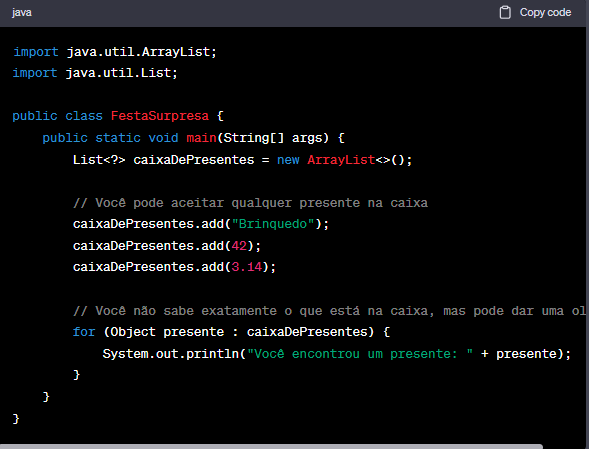
é uma colecção em que o tipo dos elementos pode ser qualquer um – tipo wildcard.

Imagine que você está organizando uma festa surpresa para um amigo, mas não tem certeza de quais tipos de presentes seus amigos vão trazer. Alguns podem trazer brinquedos, outros doces e alguns podem até trazer jogos. Para tornar a festa divertida, você deseja aceitar qualquer tipo de presente surpresa.

Em programação, os "wildcards" (ou coringas) são como essa festa surpresa. Eles são usados quando você quer criar uma parte do código que pode aceitar diferentes tipos de dados, sem precisar saber exatamente quais tipos são. Em vez de especificar um tipo de dado específico, você usa um "coringa" para representar qualquer tipo.

Existem dois tipos comuns de wildcards em Java:

<?> (wildcard não limitado): Isso representa qualquer tipo de dado. É como dizer "Aceito qualquer presente surpresa na festa!"



<? extends Tipo> (wildcard limitado): Isso permite aceitar qualquer tipo que seja um subtipo de "Tipo". Por exemplo, você pode dizer "Aceito qualquer presente surpresa, desde que seja um brinquedo."



Neste exemplo, usamos List<? extends Brinquedo> para representar a caixa de brinquedos, onde o wildcard <? extends Brinquedo> indica que a lista pode conter qualquer tipo que seja um subtipo de Brinquedo. Isso significa que você pode aceitar brinquedos (ou subtipos de brinquedos) na caixa.

Depois, adicionamos Carrinho e Boneca (que são subtipos de Brinquedo) à caixa e, em seguida, percorremos os brinquedos na caixa.

Wildcards são úteis quando você deseja criar código que seja flexível e possa trabalhar com diferentes tipos de dados, sem precisar conhecer todos os tipos possíveis com antecedência. É como abrir espaço para surpresas na programação, tornando o código mais versátil!

ArrayList

Imagine que você tem uma caixa mágica que pode crescer para acomodar uma quantidade ilimitada de brinquedos. Você pode adicionar brinquedos na caixa, tirá-los e até mesmo reorganizá-los conforme desejar. Essa caixa mágica é chamada de ArrayList.

Em programação, um ArrayList é uma estrutura de dados que funciona como uma coleção de objetos. Você pode usá-lo para armazenar uma lista de itens, como números, nomes, ou qualquer outra coisa. A coisa especial sobre um ArrayList é que ele pode crescer ou encolher dinamicamente, o que significa que você pode adicionar ou remover itens conforme necessário.

Por exemplo, você pode criar um ArrayList de números inteiros e adicionar números a ele à medida que desejar. Também pode procurar por um número específico, remover números ou até mesmo reorganizá-los.

