私の名前は，Martyn Pike です．大規模プログラムについての Ada University の講義に，ようこそ．

この講義の主題は，Ada 言語における汎用プログラムの扱いです．一緒にスライドを終えた後，学習の程度をみるための幾つかのクイズを行います．

この講義の主題は，プログラム設計やアルゴリズムの中にあるロジックのパターンを特定することにあります．

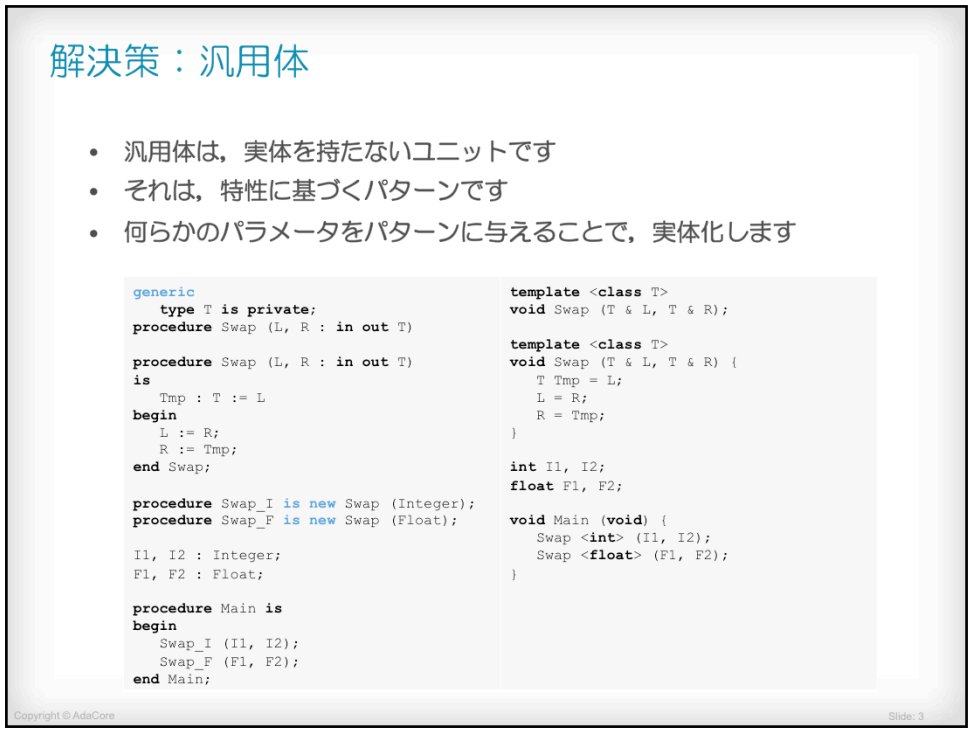
ゴールを達成するために用いているデータ型やユーティリティ副プログラムの詳細の中から，アルゴリズムの実現を抜き出すことができます．

例にあるコードの断片中で，2 つの手続きは，宣言部とパラメータの仕様が異なっていることを除けば，同一であることが分かると思います．

汎用プログラムの技には，抽象を安全に作り，アルゴリズムを実体化可能なテンプレートにパッケージ化するのは，どこかを決定することが含まれます．

このことを，最下段の擬似コード断片で示しています．手続きのボディ部は，そこで動作するデータ型の詳細を抽象化しています．この例は，整数或いはブールのパラメータ型どちらでもよく，手続きのボディ部に変更を加えることなしにコンパイルすることができます．

こういったプログラム記述の分野に関して，多くの名前があります．パターン／汎用／テンプレートに基づくプログラミングです．



Ada プログラム言語は，プログラマが，データ抽象の方法で，アルゴリズムを記述することを可能にしています．このとき，汎用体を使います．

多くの場合，Ada のソースコードは，「見たものをそのまま得ることができる（WYSWYG）」です．しかし，汎用体は，例外になります．実行時には存在しないからです．

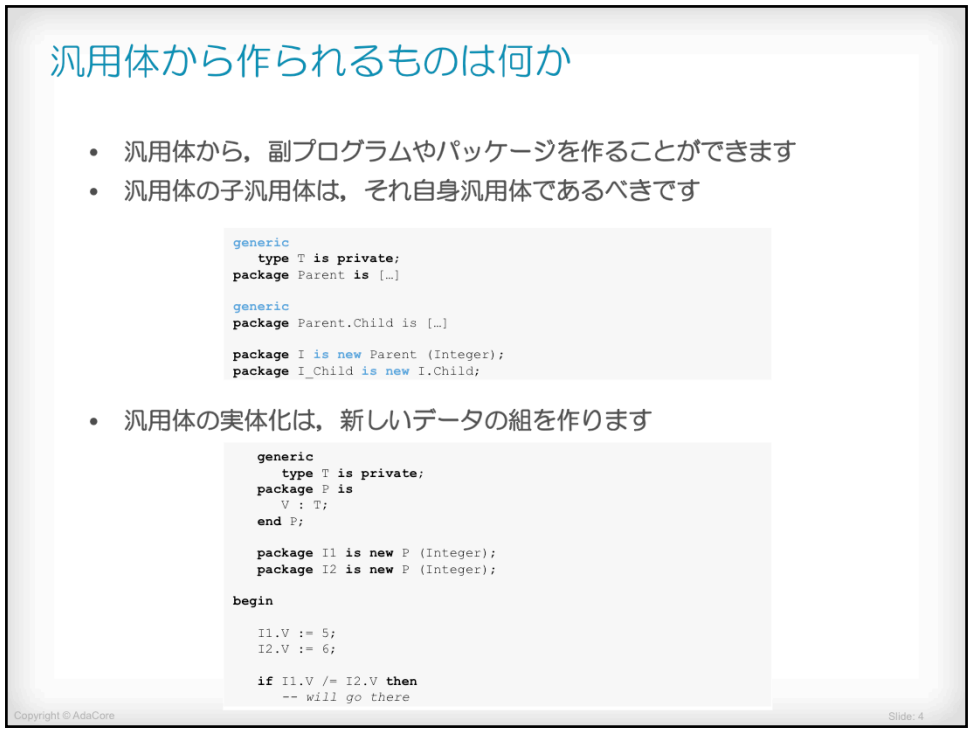
汎用体は，実行時に使うためには，開発者が実体化する必要のあるコードのことです．

本質的に，汎用体とは，抽象汎用体の実行時のコピーです．汎用体の生命は，実体化する時に与えるパラメータによって規定されます．

これは，最初は納得するのが難しい概念です．しかし，もし，この汎用体の基本を受け入れるのに十分時間を費やせば，コードの書き方が全く変わってきます．

Ada 言語のみが汎用プログラムを支援しているだけではありません．似ているものとして，C++ のテンプレートをご存じでしょう．

ここに示すコードの断片は，Ada 言語の汎用体と C++ 言語のテンプレートの仕様・実体化・利用におけるの違いを示しています．



汎用体から副プログラムやパッケージを作ることができます．

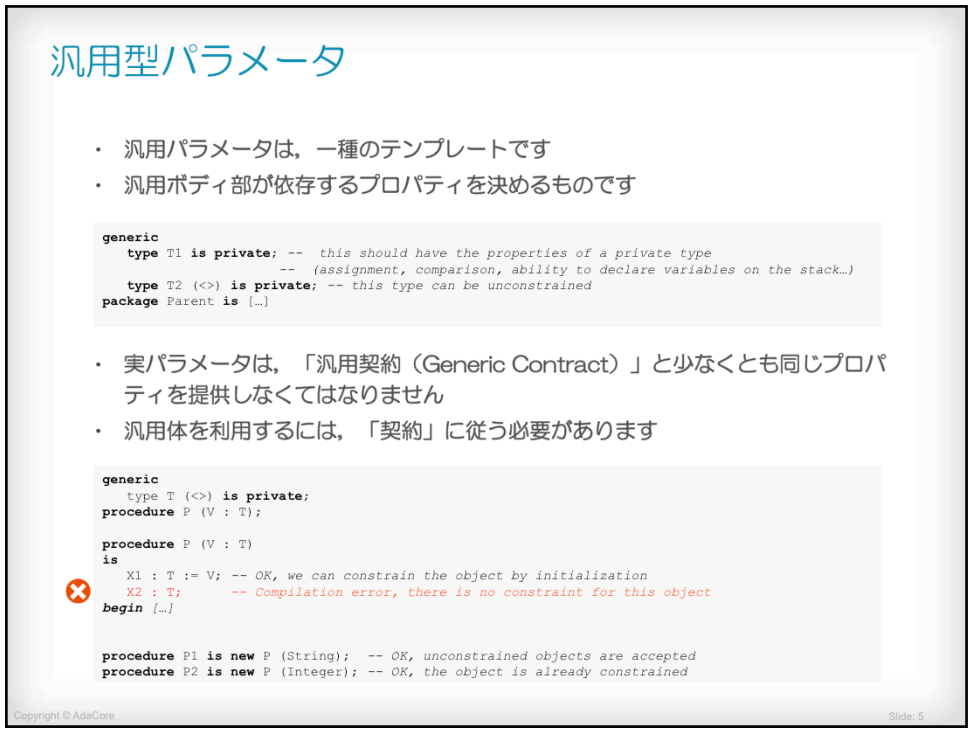
子汎用体は，それ自身汎用体でなければならないという制限があります．

このスライドの一番上のコード断片は，パラメータ T を持つ Parent という名前の汎用パッケージを示しています．また，子汎用体とみなせる Parent.Child パッケージがあります．

次に，親汎用パッケージと子汎用パッケージを実体化しています．

重要なのは，各汎用体の実体化は，完全に新しいデータの組を作り，各実体のデータ間には，何の関係もないということです．

公開された変数 V を持つ汎用パッケージを用いて，例を示します．各 P の実体は，I1 と I2 において，異なったバージョンの V を持ちます．（等号否定）比較演算子は，真と評価します．2 つの異なった記憶領域で代入を行っているからです．



アルゴリズムのどの部分を「抽象」とし，どの部分を「パラメータ」とするかは，汎用体のプログラムで重要な部分です．

汎用体の仕様部では，パッケージのユーザに対して，実体化においていかにパラメータ化がなされるか，ということについての明確な文を提供しなければなりません．これは，パッケージの利用者と汎用アルゴリズム間の契約として働きます．

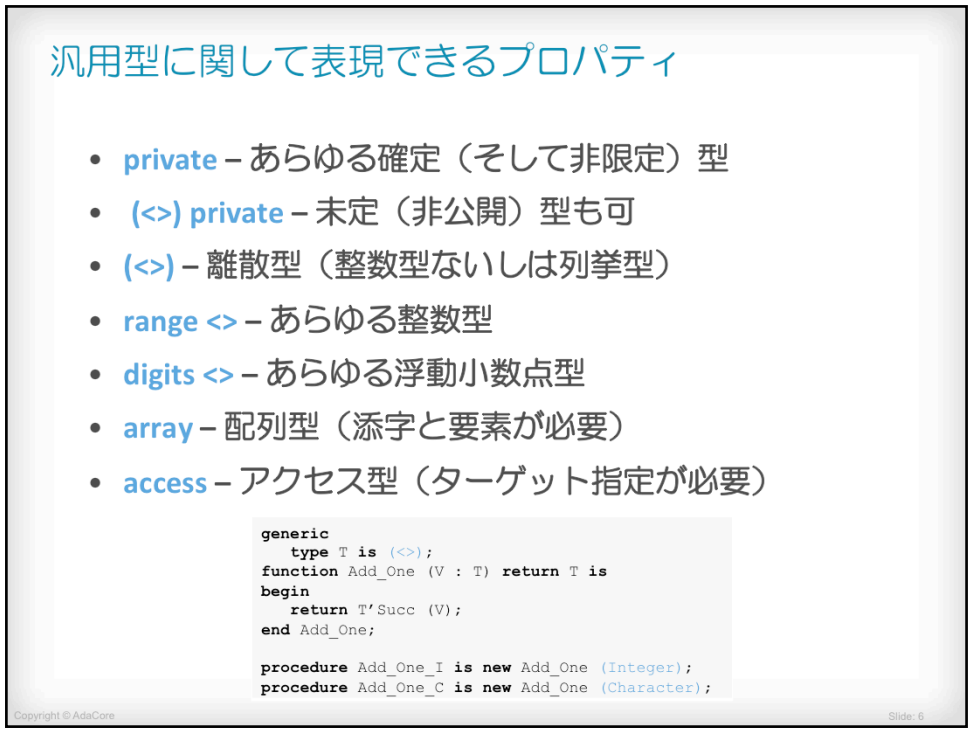
最初のコード断片は，次のことを示しています．型 T1 は，非公開型のプロパティを持たなくてはならず，T2 は無制約でなくてはならないということです．

汎用体中で，実際のパラメータとして何が提供されても，実体化では，少なくとも汎用契約と同じだけのプロパティを提供する必要があります．

汎用アルゴリズムは，パッケージの利用者が，「契約」をどれだけ尊重するかに依存しています．これは，コンパイル時の Ada の型システムによって，強制されます．逆の立場から見ると，汎用アルゴリズムもまた，契約を尊重しなくてはなりません．

二番目のコード断片は，型 T が用いられた時に，汎用契約の仕様は，型は無制約であるとは述べていますが，制約が与えられていないということを，コンパイラが如何に見つけ出すかを示しています．

コンパイラと Ada 型システムは，パッケージ利用者と汎用体が契約に適合していることを保証します．



契約における全ての型のプロパティは，汎用体のユーザとコンパイラに対して，重要な情報を提供しています．

ここには，汎用体の作成者が利用可能なプロパティを示しています．

まず一番最初に，非公開 (private）型を必要とする汎用体の契約を書くことは可能と云うことをお伝えします．ここには，非限定単純非公開型と未定非公開型があります．

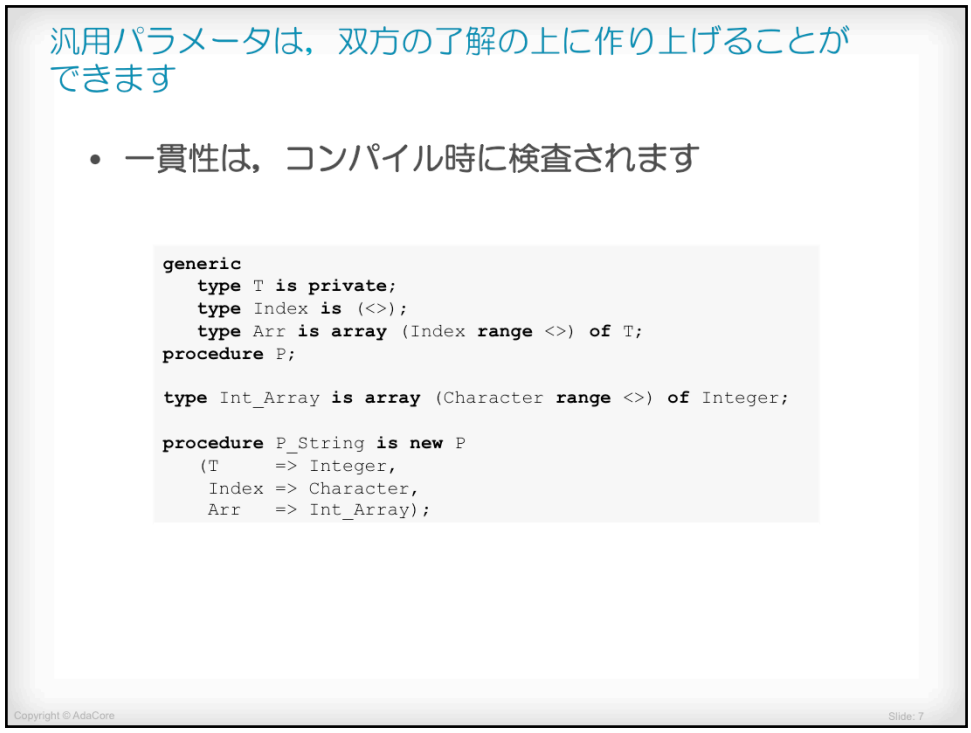
単純非公開型と未定非公開型の違いについての説明は，このシリーズのカプセル化の講義にあります．

整数型・列挙型を含む全ての離散型のプロパティを用いて，汎用型を記述することが可能です．

純粋な整数型・浮動小数点型・アクセス型を使用できます．

スライドの最後にあるコードの断片は，汎用関数の例を示しています．これは，何らかの離散型，例えば整数，文字型によって実体化することができます．

配列型を飛ばしたことにお気づきでしょう．次のスライドで説明します．



添字と要素型とともに，配列を汎用契約中に全て記述することが可能です．

このことで，汎用契約が互いの上に作り上げる型を含むことができるようになります．

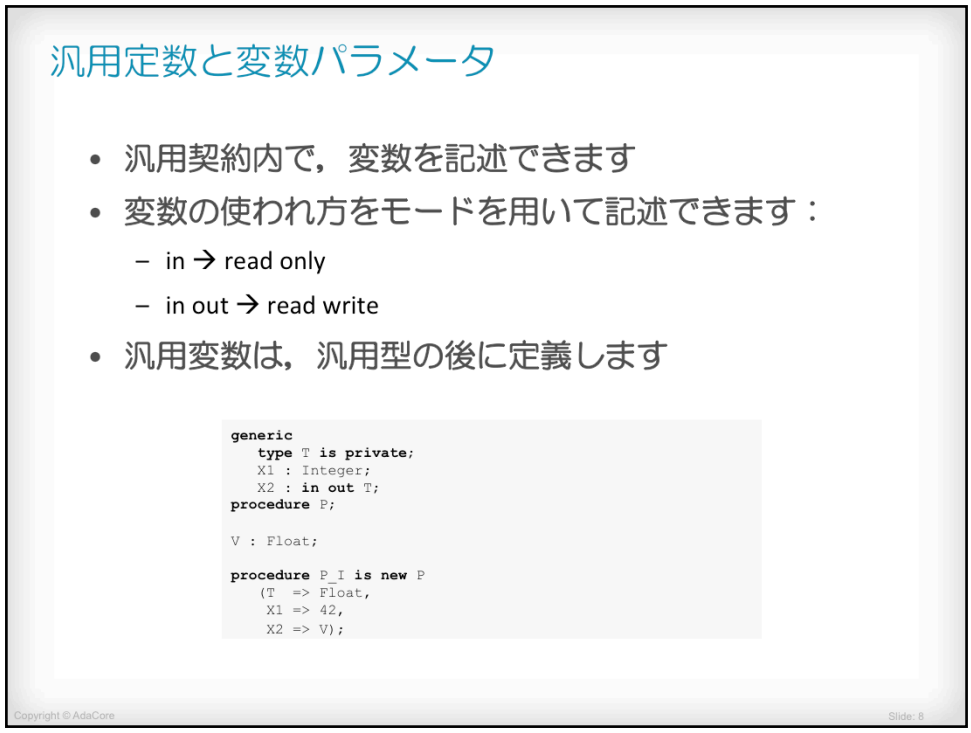
例には，P という名前の汎用手続きがあります．最初の契約は，単純非公開型 T を持っていることを示しています．次に離散型の Index があります．

次に配列型 Arr の要素型として T　を用いています．Index は，配列の添字です．

例では，P を手続き P_String として実体化しています．このとき，T に対しては整数を，添字には Index を，Int_Array 型には Arr を用いています．

ここで理解すべき重要な点は，この層別における一貫性は，コンパイル時にコンパイラにより検査されるということです．

汎用体を実体化するにあたり，名前付きパラメータをもちいることは，契約を尊重していることを確認するために重要である，ということを指摘したいと思います．



次に，汎用型を考えます．更に，汎用契約中に現れる変数パラメータを見てみます．

最初に，手続きや関数のパラメータと同様のモードを用いる変数を取り上げます．モードとは即ち，「in」，「out」，「in out」です．

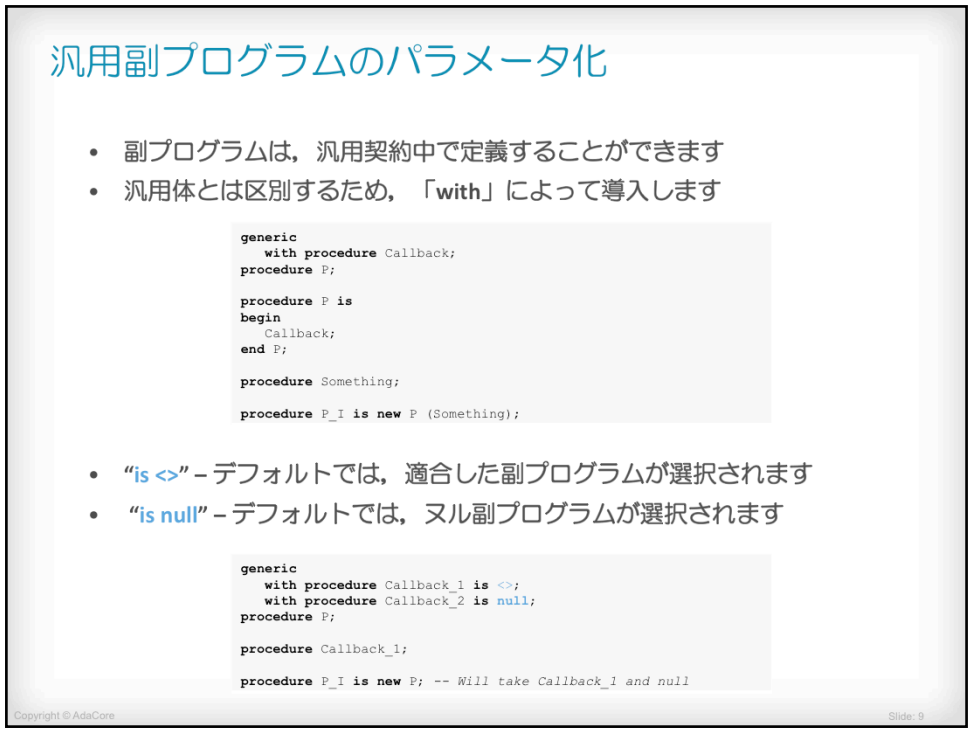
定数には，「in」変数を用います．

これらのパラメータは，利用者に対して，汎用体は，（関数ないしは手続きのように）呼び出し時ではなく，実体化時にオブジェクトを提供するように求めていることを示しています．

先に配列型を用いた汎用パラメータについてみたように，汎用変数を他の汎用型で用いることができます．

スライド下部の例において，P は，非公開型 T の契約を持ちます．また整数と 「in out」 非公開型変数を持ちます．

コンパイラは，P が P_I として実体化されたときに，変数 X2 は，T に対して記述されたものと同じ型であることを保証します．



最後に，汎用パラメータ型による副プログラムの利用法を見ます．

汎用副プログラムパラメータは，汎用契約中で定義する必要があります．「with」予約語を先頭に使います．これによって，汎用体自身が関数や手続きといった副プログラムである，ということに関し混乱を生じさせないためです．

例として，契約中に Callback という名前の手続きを持つ汎用手続き P について考えます．P のボディ部は，その機能を完了させるために，Callback を呼び出さないといけないとします．

P の実体化において，Something 副プログラムは，汎用パラメータ Callback として提供されます．

もしパラメータが実体化時点で指定されていない場合に，用いるデフォルトを記述することもできます．

デフォルト値は，名前で適合した副プログラムないしは，ヌル副プログラムです．

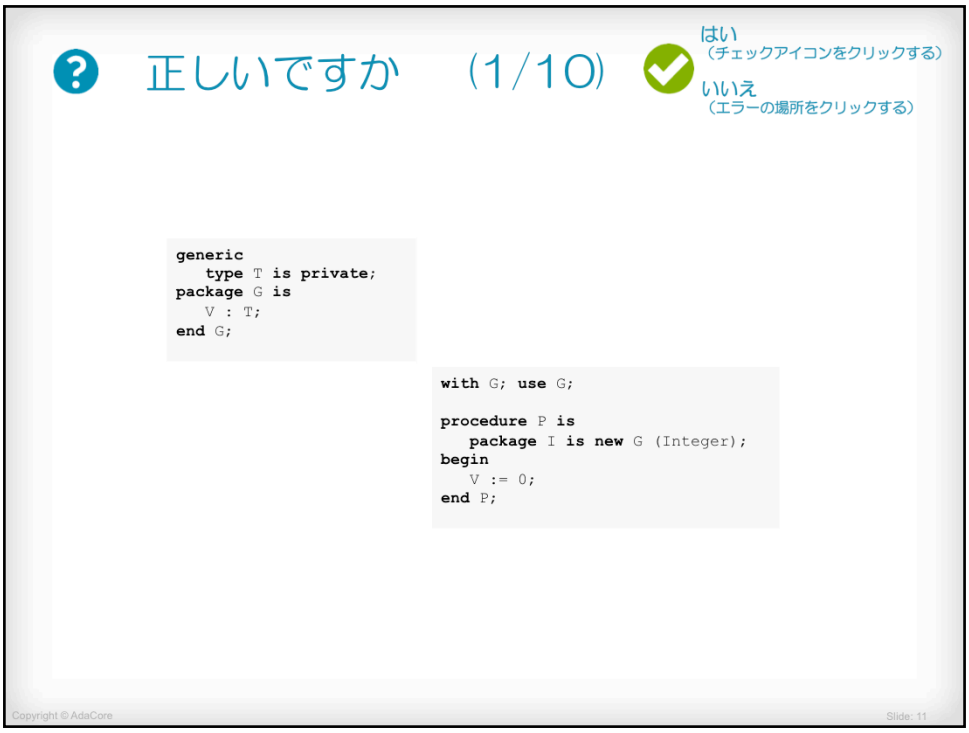
このスライドの最後の例は，デフォルトの実体化を示しています．ここでは，Callback_1を名前から選択し，Callback_2に対してはヌル副プログラムを選択します．

次のことを繰り返したいと思います．コンパイラは，汎用契約の全ての側面における一貫性を検査できるということです．

講義の最後のセクションとなりました.

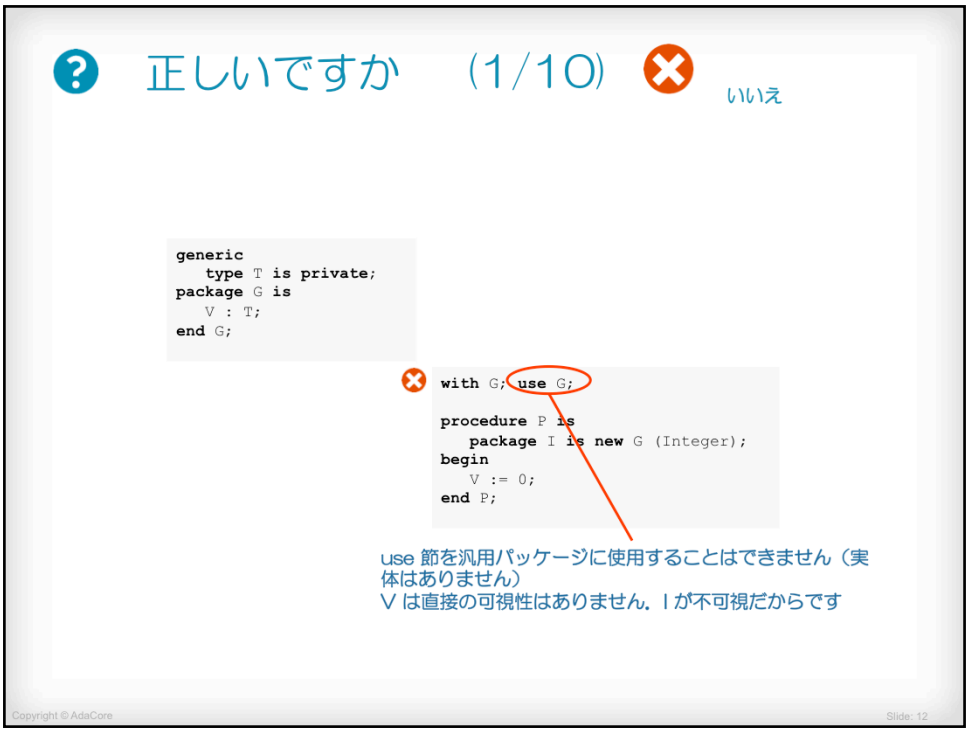
もう，Ada における汎用プログラムについて十分な知識を持っていることと思います．あなたの理解を確認するように設計された質問からなるちょっとしたクイズをやってみましょう.

幸運を！



最初の問題は，汎用パッケージ G の仕様部から始めます．この仕様部は，非公開汎用パラメータ T 型を持っています．手続き P は，G を仮パラメータとして整数型で実体化しています．

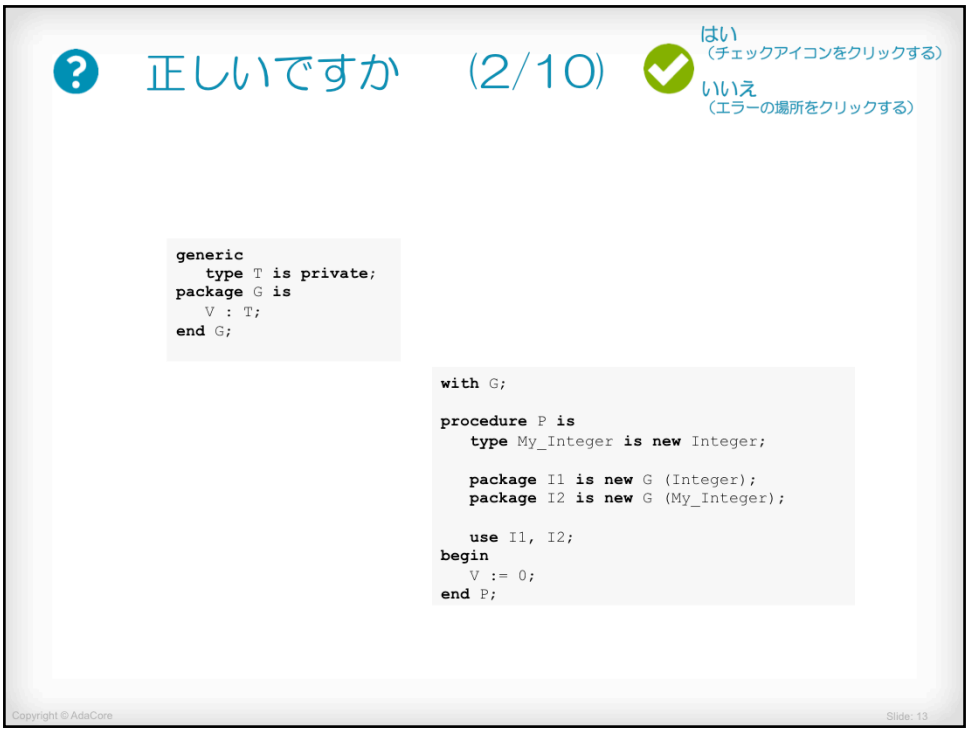
このコードが正しいと思うときは，「はい」のアイコンをクリックしてください．間違いと思えば，その場所をクリックしてください．



このコードは間違いです．幾つかの理由からコンパイルに失敗します．

use 節は，汎用パッケージに用いることはできません．実体がまだないからです．

二番目として，V には可視性がありません．実体 I には use 節がないからです．

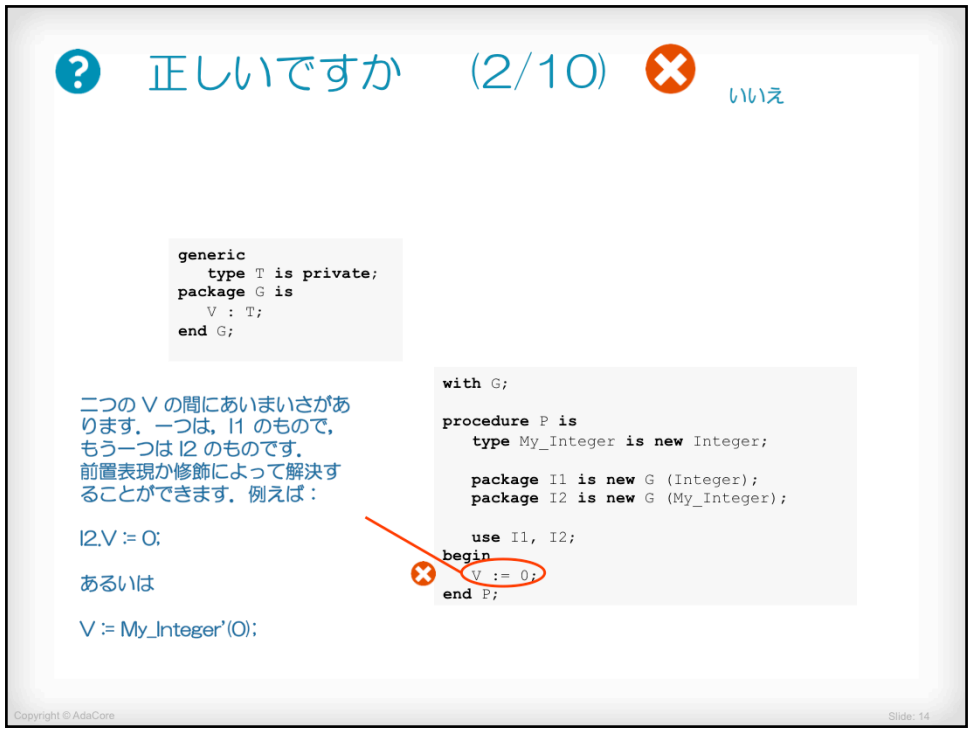


次の問題は，前回と同じ汎用パッケージ G の仕様部を使います．右側は少し異なっています．手続き P で，use 節を使用しません．

My_Integer 型は，整数型から派生しています．この型をパッケージ I2 として，G を実体化するときの仮パラメータとします．パッケージ I1 として実体化するときは，単純に整数型を用いています．

両方の実体に関して use 節を使用しています．

このコードが正しいと思うときは，「はい」のアイコンをクリックしてください．間違いと思えば，その場所をクリックしてください．



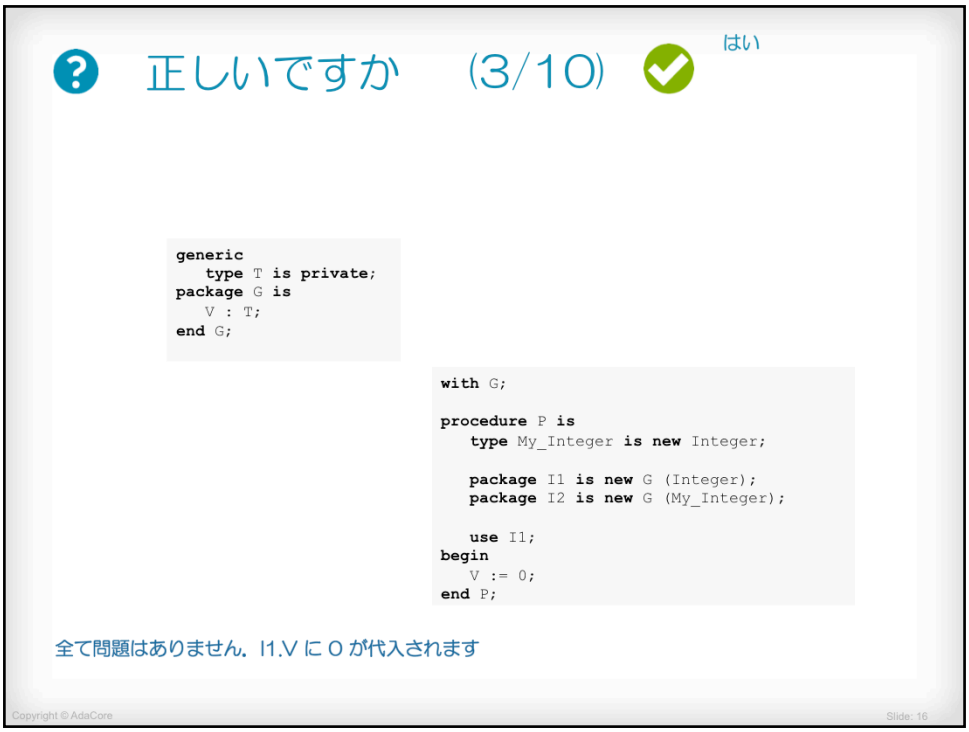
このコードは正しくありません．v を参照していますが，どちらのパッケージの v かがあいまいで解決できないためです．

このコンパイルエラーを解決するための一つの方法は，パッケージの名前を v の前に修飾として付けるか，代入する値に型修飾を付けるかです．これによって，コンパイラはその型に基づきパッケージを決定できるようになります．

問題３では，再度汎用パッケージの同じ仕様部を用います．

今回，手続き P は，G を My_Integer で実体化した I2 が，use 節に含まれていません．

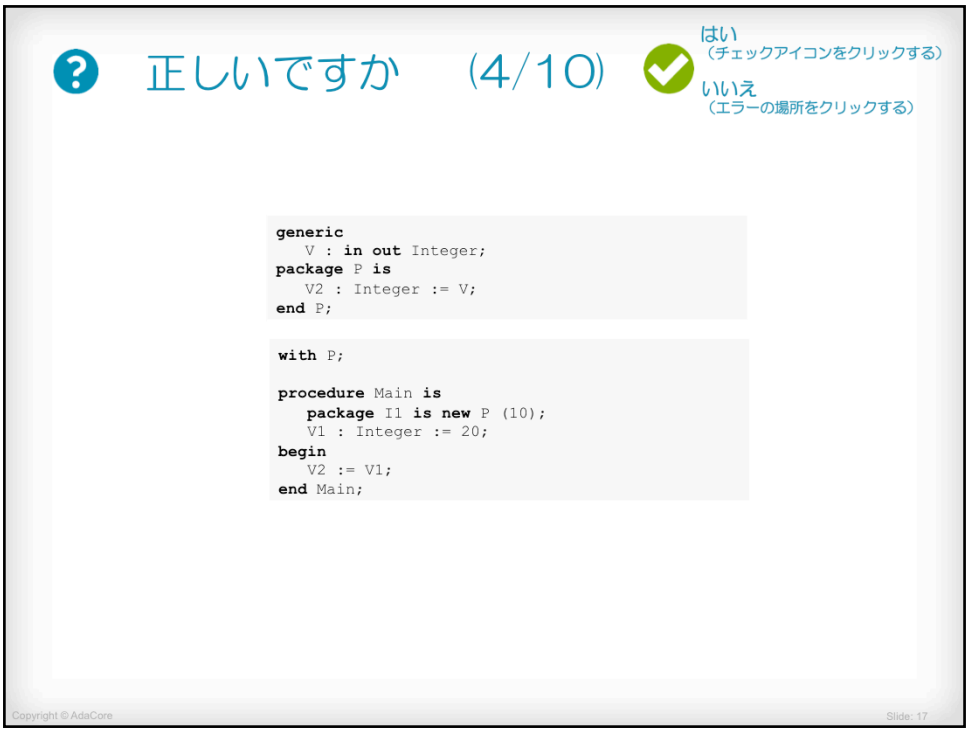
このコードが正しいと思うときは，「はい」のアイコンをクリックしてください．間違いと思えば，その場所をクリックしてください．



今回，コードは正しいです．エラーなしに，コンパイルできるでしょう．

コンパイラは，Vが I1 に属していると分かります．I1 に対してのみ use 節があり，I2 にはないからです．

これは，Ada プログラム言語の強い型づけモデルによってのみ達成可能です．



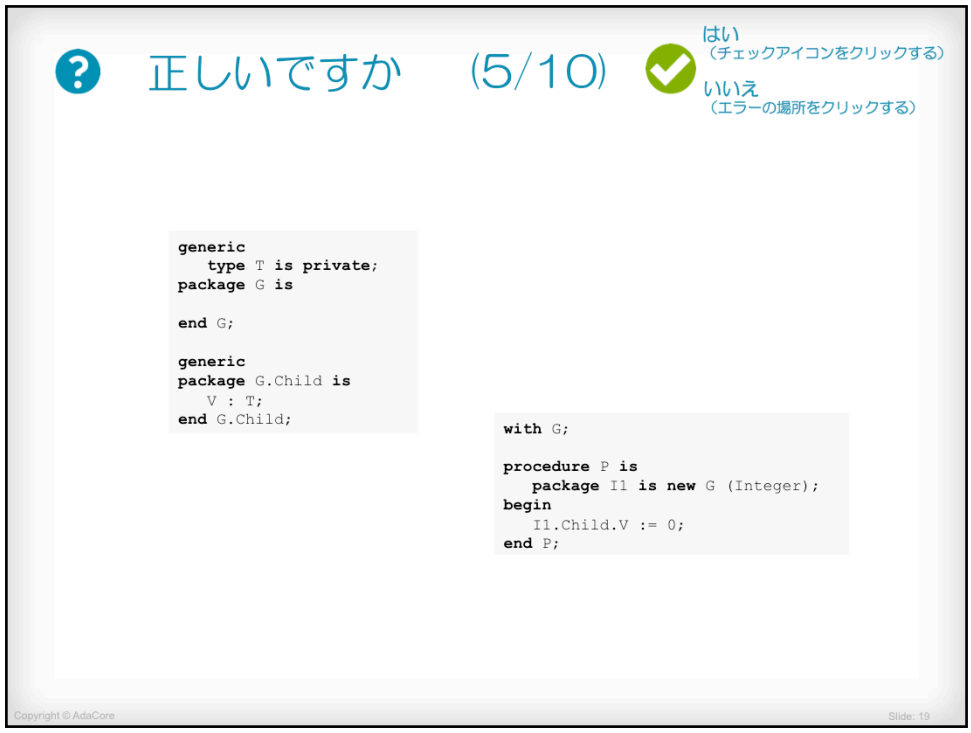
次の問題では，新しい汎用パッケージ仕様 P を用います．契約として，読み書き可能な変数 V と，公開されている整数変数 V2 を，初期化時に使用するパラメータの値としています．

手続き Main は，P を I1 として実体化します．汎用契約変数として，数字 10 を使います．

このコードが正しいと思うときは，「はい」のアイコンをクリックしてください．間違いと思えば，その場所をクリックしてください．

コードは誤っており，コンパイルできません．

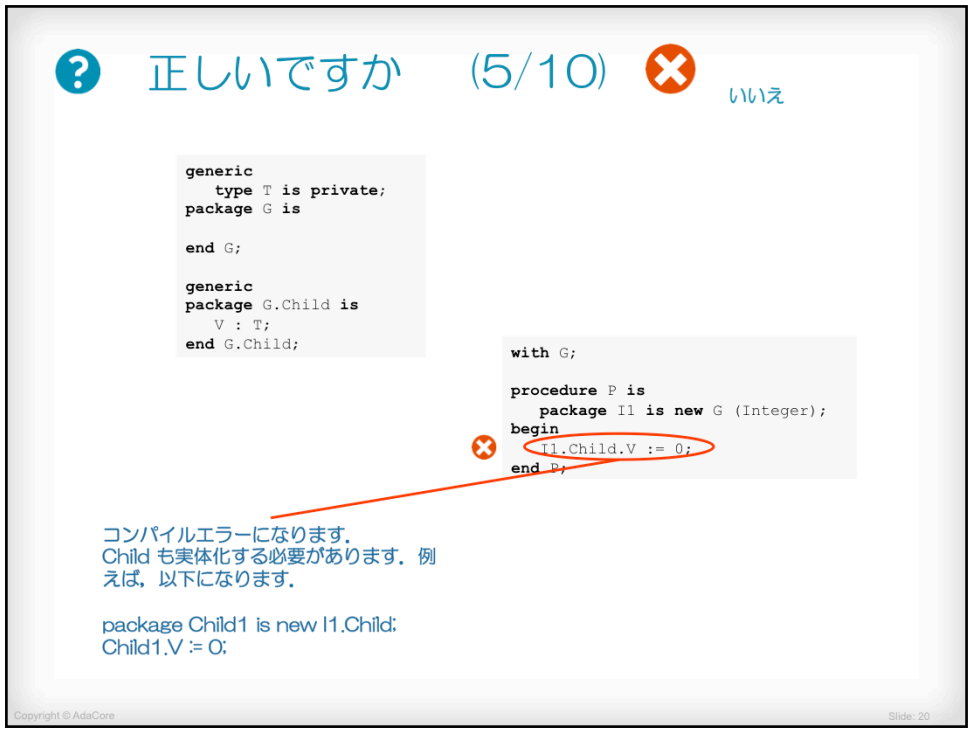
汎用パッケージ P の仕様部では，変数 V は，「in out」であり，実体化したものはすべて，引数として読み書き可能な変数が渡されることを保証しなくてはなりません．



次の問題では，汎用パッケージ G と，G の子汎用パッケージ Child を使います．

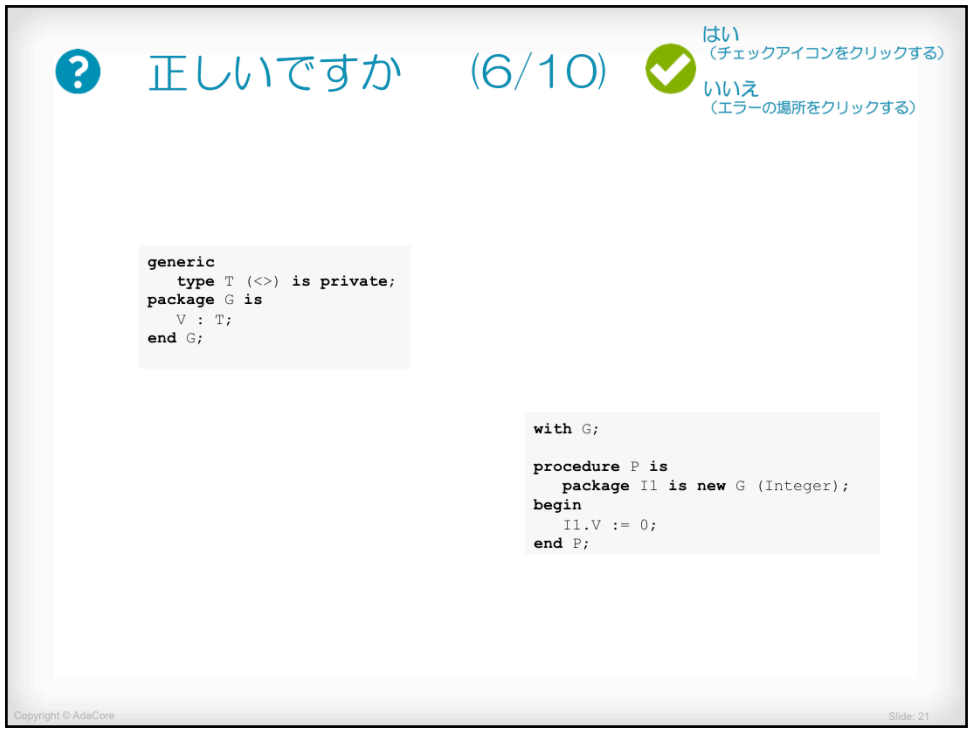
手続き P 中で，汎用パッケージ G を I1 として，整数型をパラメータとして与え，実体化しています．次に，G.Child パッケージの変数 V にアクセスを試みています．

このコードが正しいと思うときは，「はい」のアイコンをクリックしてください．間違いと思えば，その場所をクリックしてください．



このコードは正しくありません．コンパイルを行うとエラーになります．

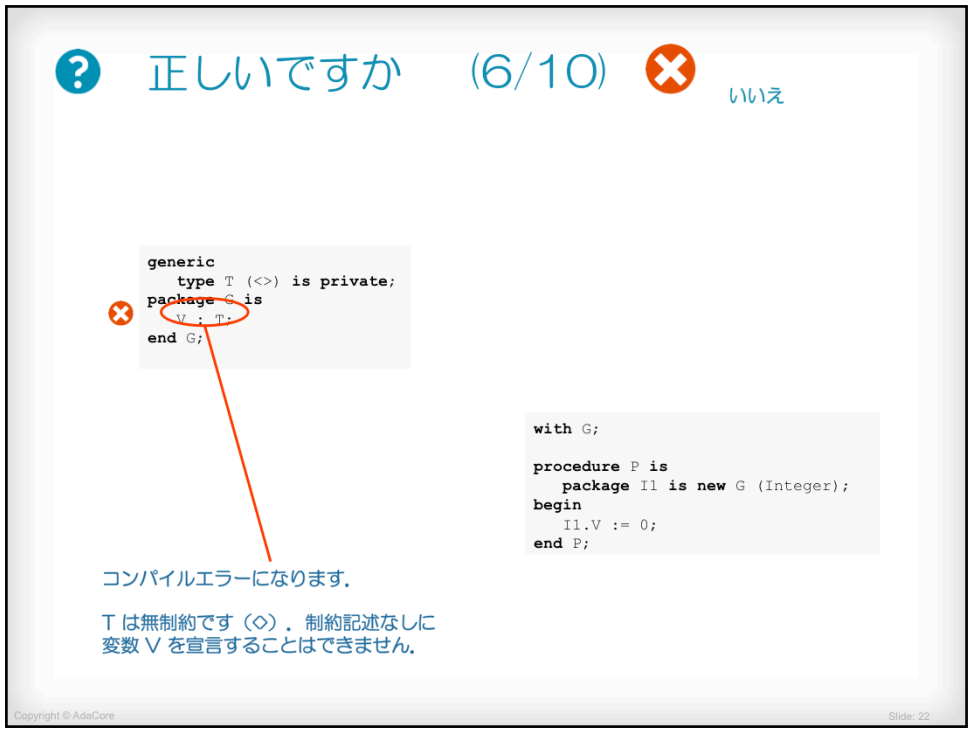
汎用子パッケージは，（親パッケージとは別に）実体化される必要があります．例えば，Child を Child として実体化し，その後，変数 V にアクセスします．



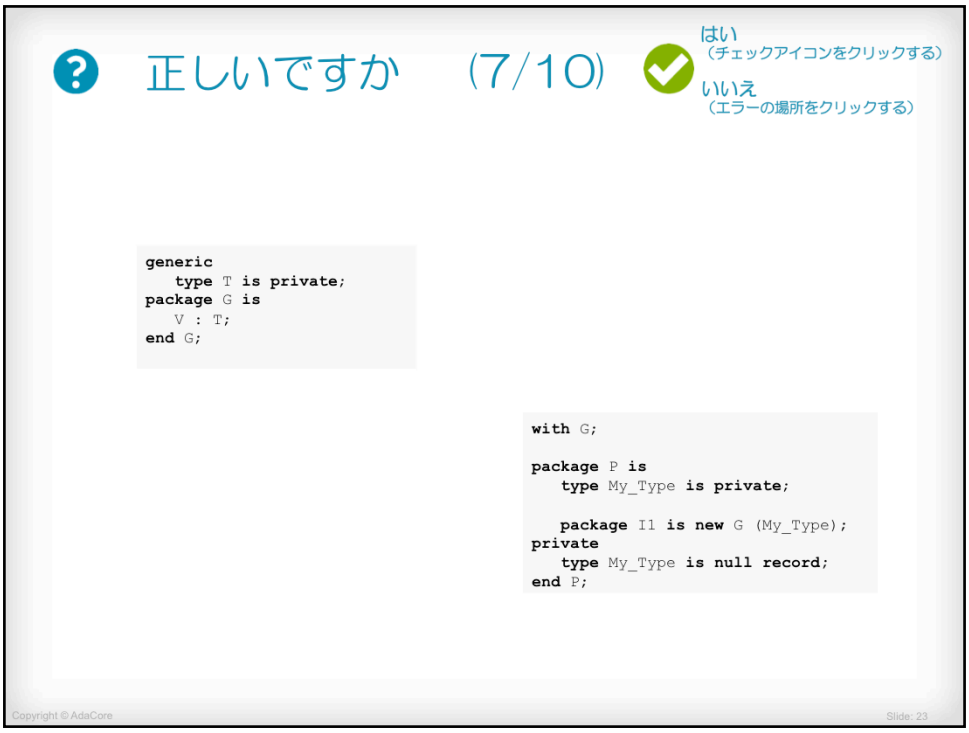
次に，汎用契約中で，未定非公開型 T を含む仕様部を持った汎用パッケージを用います．

この手続き P は， G を整数引数とともに実体化し，パッケージの変数 V にアクセスします．

このコードが正しいと思うときは，「はい」のアイコンをクリックしてください．間違いと思えば，その場所をクリックしてください．



このコードは正しくありません．T は未定型であるためコンパイルに失敗します．汎用パッケージの実体化の時点で，制約をつけなければなりません．

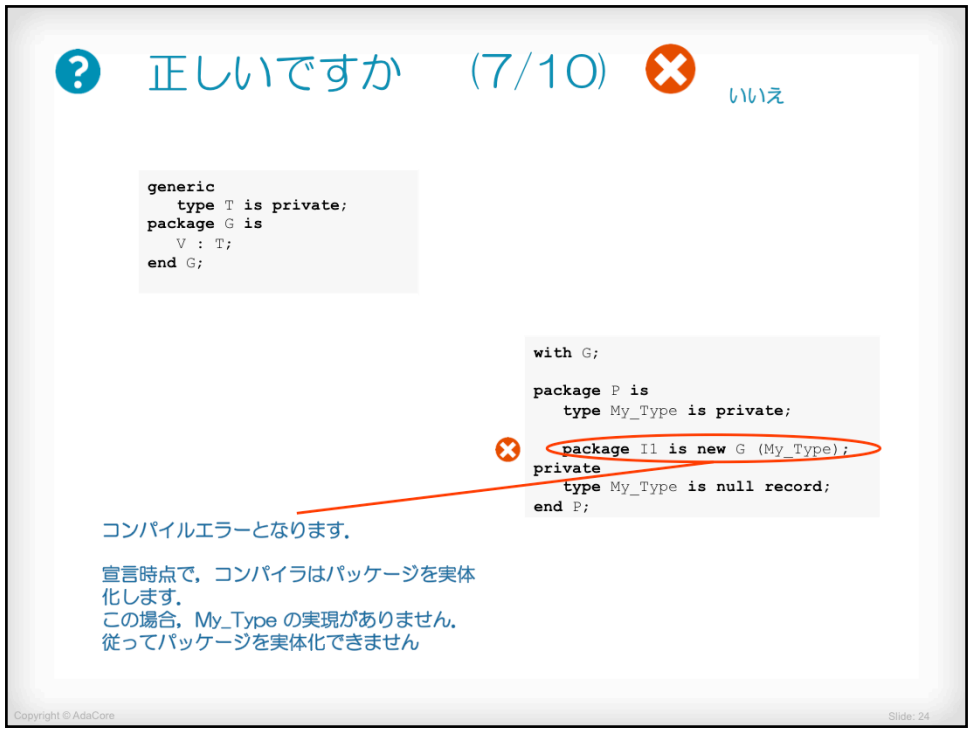


問題 7 は，汎用パッケージ G に対して，汎用パラメータとして単純な非公開型を用いています． T は，G の実体の中では，公開変数としてして用いられています．

パッケージ P は，非公開型 My_Type を宣言しており，汎用パッケージ G を実体化するために用います．

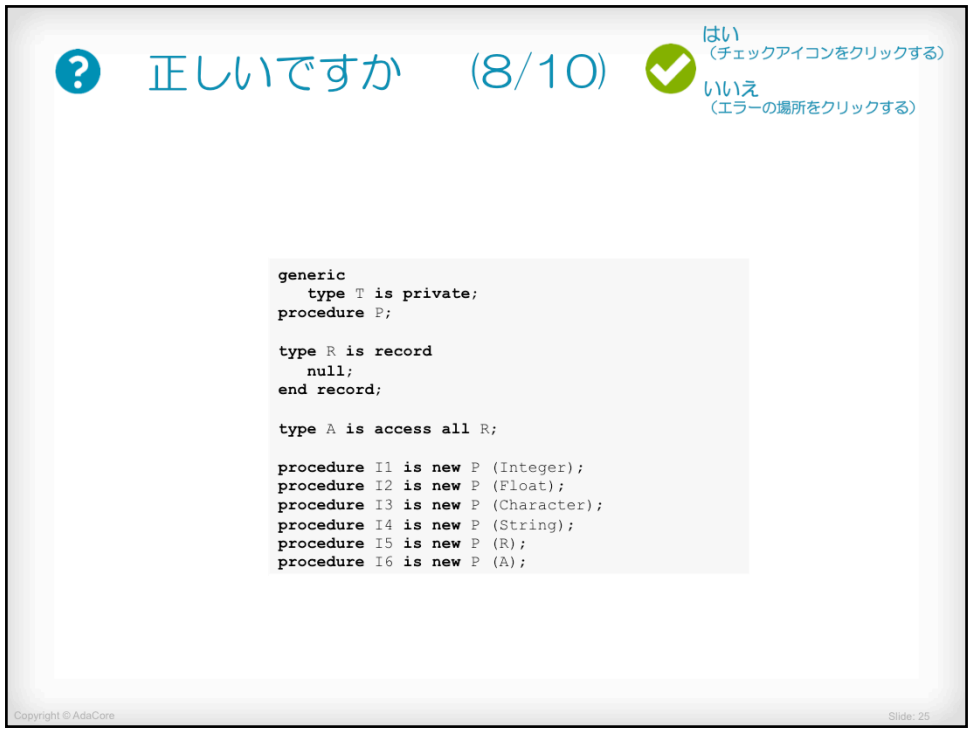
P の非公開領域では，My_Type をヌルレコードとして実現しています．

このコードが正しいと思うときは，「はい」のアイコンをクリックしてください．間違いと思えば，その場所をクリックしてください．



このコードは誤っています.

コンパイルは失敗します．My_Typeは，引数としてパッケージ G を I1 として実体化する時点では，実現されていないからです.

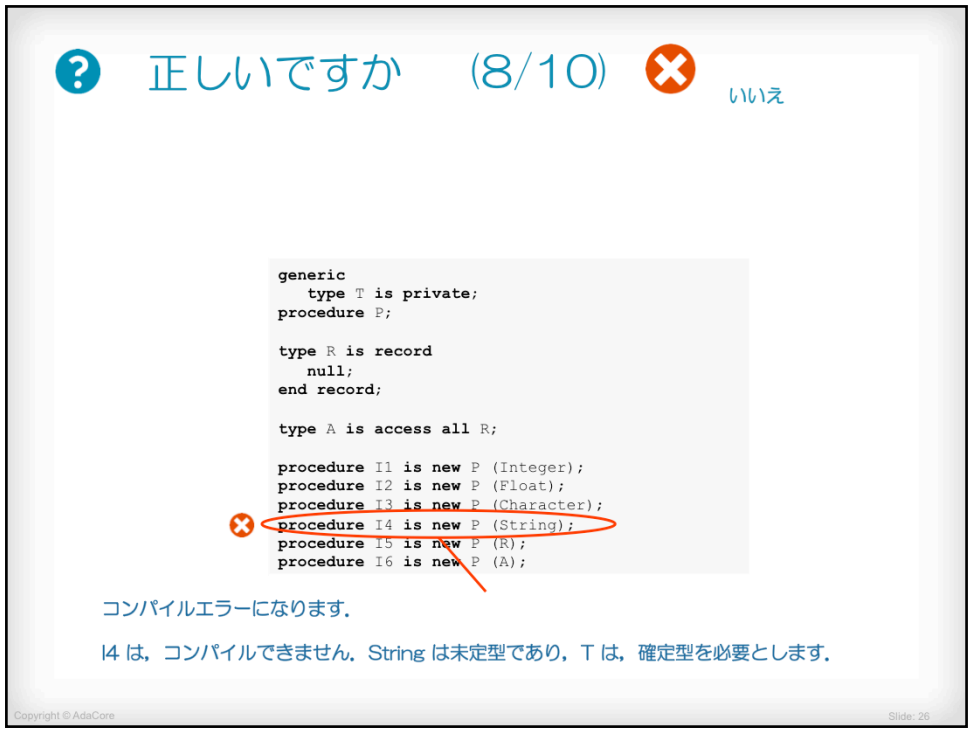


次の問題では，汎用手続き P を定義します．この P は，実体化時に単純非公開型 T を必要とする契約を持ちます．

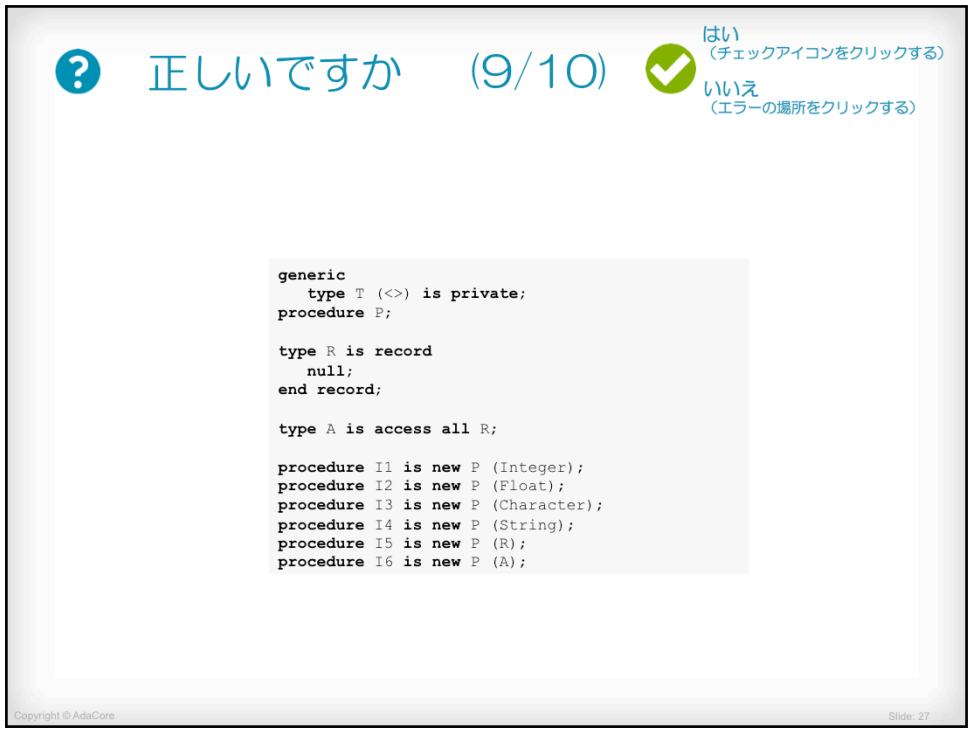
レコード型 R があります．その記憶域プールないしはスタック領域にアクセスするために，アクセス型 A を宣言します．

P を異なる引数で，何度か実体化します．

このコードが正しいと思うときは，「はい」のアイコンをクリックしてください．間違いと思えば，その場所をクリックしてください．



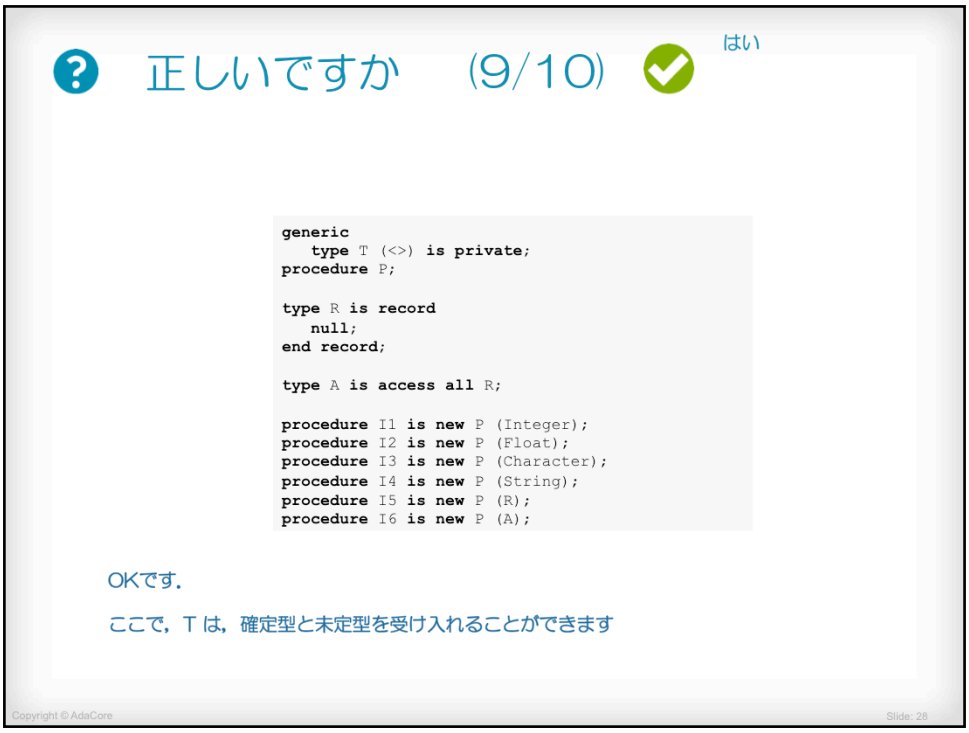
このコードをコンパイルしようとすると，コンパイルエラーが発生します．String は，未定型であり，P に対する汎用契約は，確定単純型を要求しているからです．



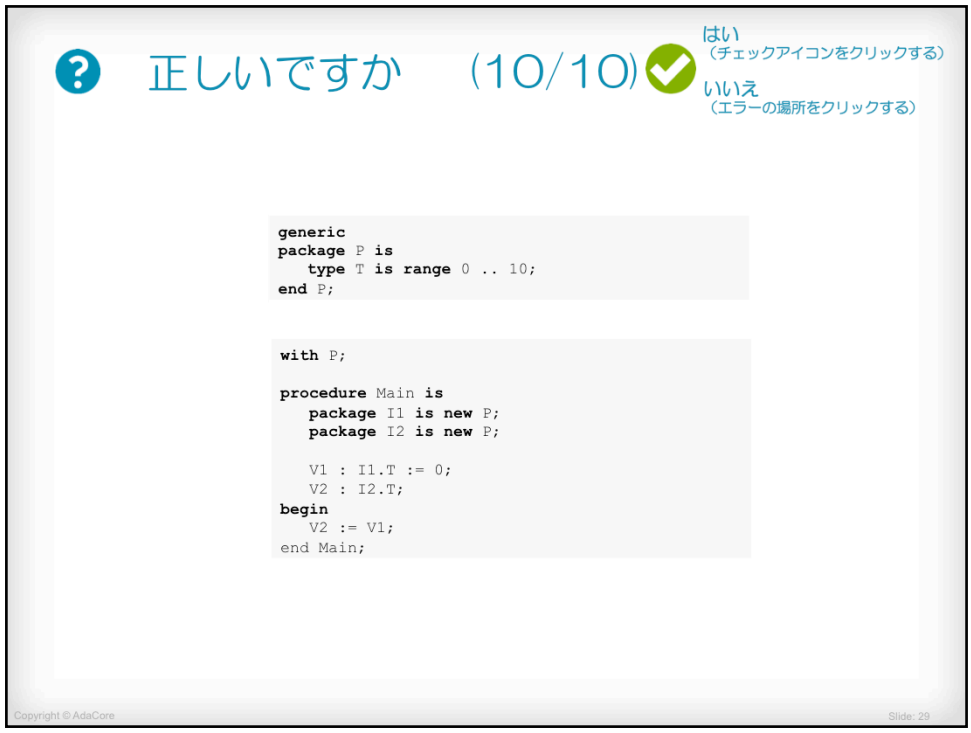
最後から二番目の問題は，前回の問題に似ています．ただし，今回手続き P の汎用契約は，未定非公開型を必要としています．

前回と同様の実体化を行っています．

このコードが正しいと思うときは，「はい」のアイコンをクリックしてください．間違いと思えば，その場所をクリックしてください．



このコードは問題ありません．完全にコンパイルできます．汎用パラメータ T は，確定型と未定型を受け入れることができます．

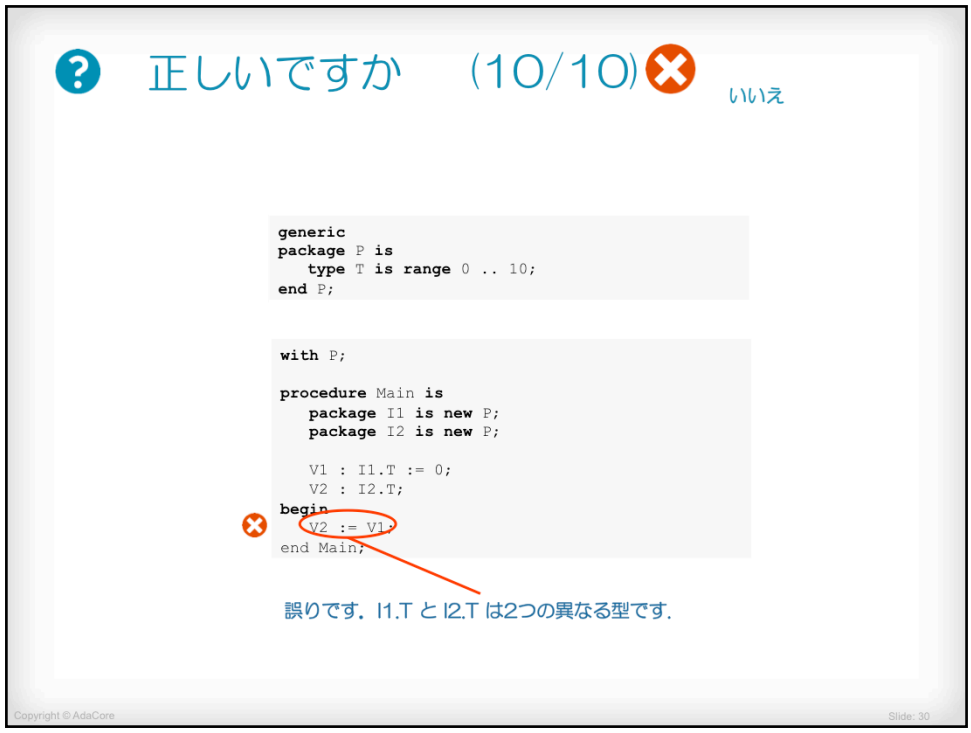


最後の問題です．汎用パラメータを持たない汎用パッケージがあります．新しい型 T を 0 から 10 の範囲として公開で宣言しています．

Main 手続きでは，二つの P の実体化を行い I1 と I2 としています．次に，二つのスタック領域オブジェクトを， I1 と I2 の公開型 T を用いて作っています．

次に，オブジェクトの一方に，他方を代入しています．

このコードが正しいと思うときは，「はい」のアイコンをクリックしてください．間違いと思えば，その場所をクリックしてください．



このコードは間違っており，コンパイルに失敗します．I1.T と I2.T は完全に異なる別の型であり，互いに相容れません．

大規模プログラムの一連の講義における Ada の汎用性に関するコースに参加いただきありがとうございました.

Ada プログラム言語を学ぶ上で，この講義が貴重なーステップとなったこと，Ada University における他のコースも引き続き受けられることを期待しています.

ありがとうございました.