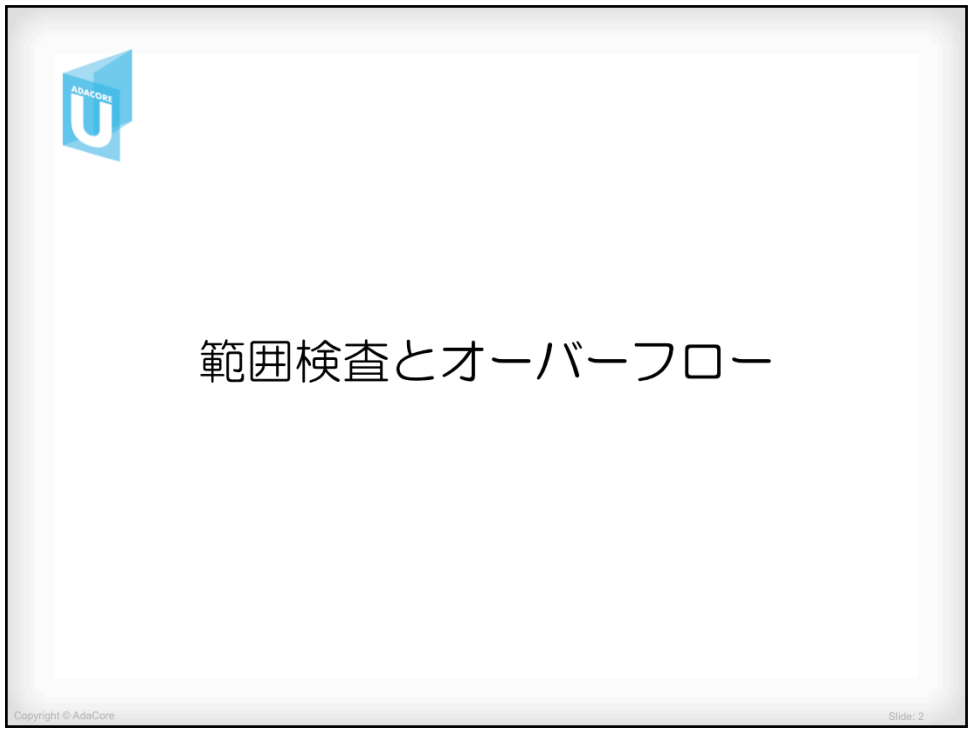
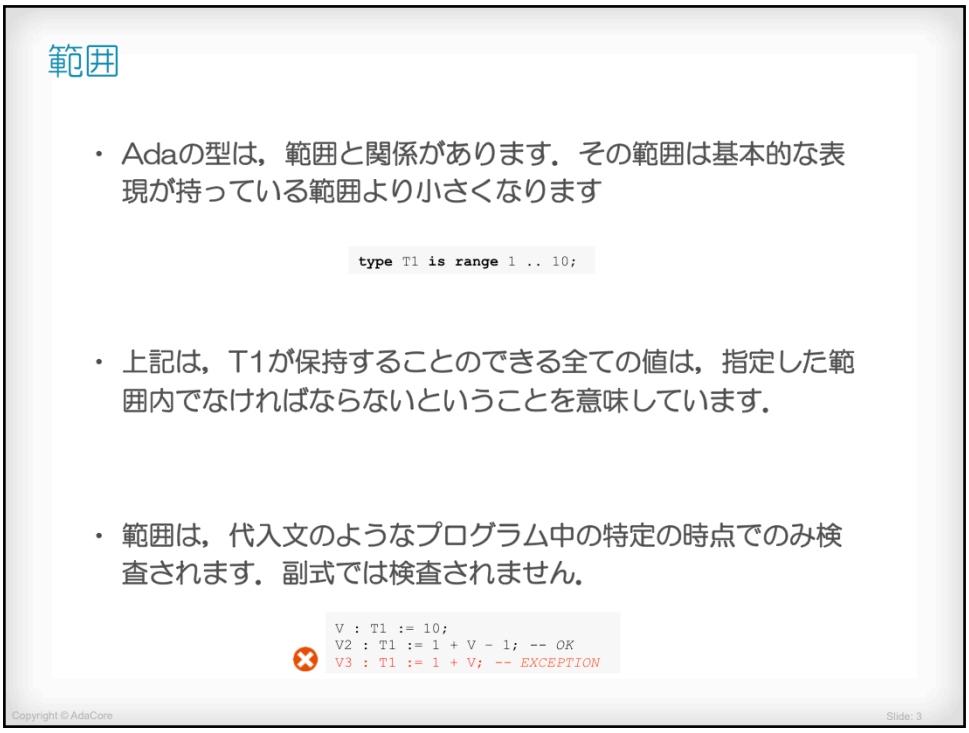
私の名前は，Martyn Pike です．大規模プログラムについての Ada University の講義に，ようこそ．

この講義の主題は，Ada 言語における型づけの問題です．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．

スライドは，3 つのセクションに分かれています：範囲検査とオーバーフロー・型基本操作・派生型です．



この講義の最初のセクションでは，Ada 言語における範囲検査とオーバーフローの基礎を紹介します．



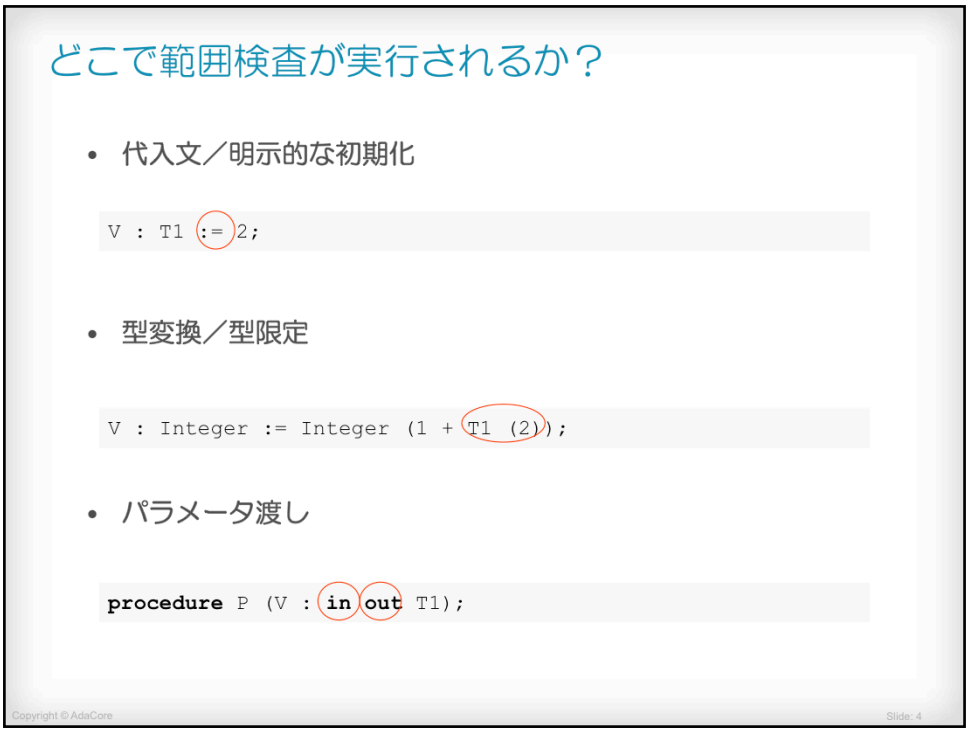
全ての高レベルプログラム言語は，最終的に，メモリ・中央処理装置（CPU）のレジスタを使うマシン命令にコンパイルされることをご存じでしょう．

レジスタとメモリの場所は，Ada 型の基本的なデータ表現です．しかし，それらが，二値化した中身のコンテキストを，知ることはありません．

実際に，多くの場合，プログラム中の Ada 言語における型は，計算機の基本的なデータ表現の範囲よりも，より小さい範囲となります．

ここでは T1 型を紹介しています．1 から 10 までの範囲を持ちます．T1 の全てのオブジェクトは，その範囲で，値を持たなければなりません．

値は，代入のようなプログラムフローのある時点で，範囲に関する検査を受けます．副式では，検査されません．最後のコード断片は，いつ検査が生じるかの例となっています．

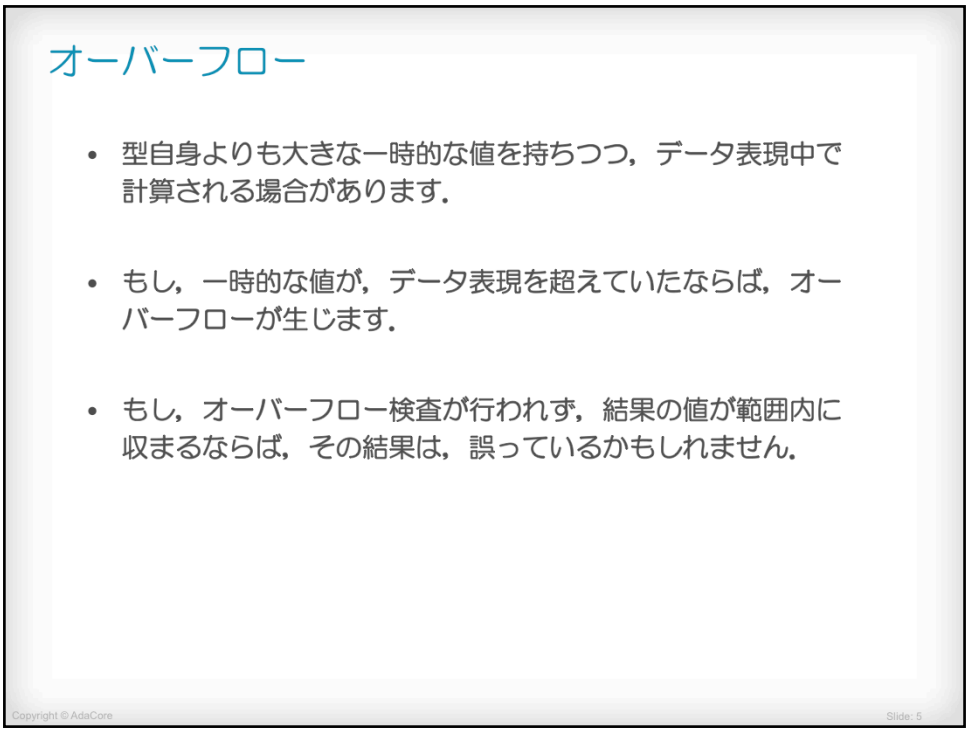


プログラムフロー中の，どこで範囲検査行われるかに関して，より具体的な例を幾つか示します．

明示的な初期化の時に実行されるものを含み，あらゆる種類のオブジェクトの代入を行う場合，その型の範囲が検査されます．

ある型から別の型への型変換，或いは，型に対する値の限定を行うとき，範囲が検査されます．

最後に，手続きないしは関数に，パラメータとしてオブジェクトを渡すときにも，範囲検査を行います．

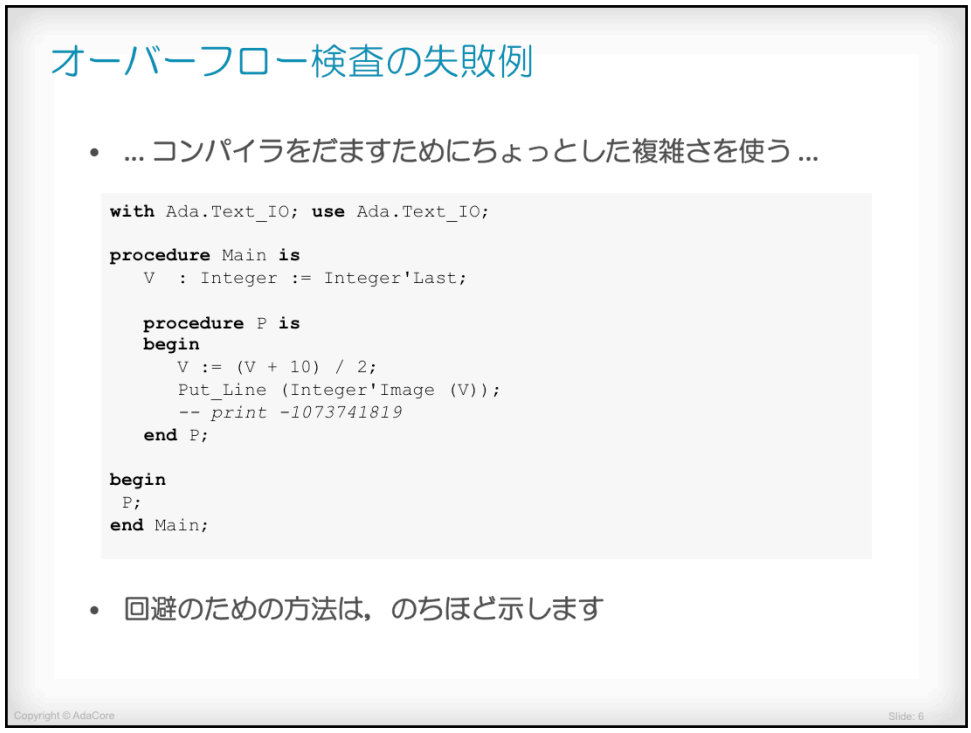


オーバーフローによって引き起こされる潜在的な落とし穴について，次に考えてみます．

高級言語では，プログラムフローは，メモリと CPU レジスタを用いて一時的な結果を得ます．その値は，計算機の表現内ですが，オブジェクトの型からたまたま外れているかもしれません．

もちろん，型が，データ表現を超えていれば，ハードウェアオーバーフローが生じます．

しかし，もし，オーバーフローが生じることなく，計算結果が最終的に範囲内に収まる場合，その結果はエラーを含む場合があります．これは，大きな混乱を引き起こします．プログラムを実行する度にしばしば計算結果が異なり，デバッグはほぼ不可能になります．



ここでは，ちょっとしたコードが，如何にコンパイラをだまし，オーバーフローを見落としと，誤った計算をさせるかということを示すために，作為的に作った例です．

このコード断片を実行すると，副式 （V+10） は，整数の正当な範囲を超えた一時的な結果となります．V はすでに明示的に整数範囲で許されるもっとも大きな数で初期化されています．従って，10 を加えることで，オーバーフローを引き起こします．しかし，これは副式の中で生じ，代入を行うときには半分の値となるため，範囲内に収まります．

範囲検査は，プログラムフローが，副式を実行しているときには，範囲検査は行われないということを思い出しましょう．

V に代入したときに，その値は，整数の正当な範囲内にたまたま収まっています．

ここでは，例外は生じません．しかし，型システムは，だまされています．

のちほど，こういった状況を避ける方法について，示します．

Ada の型は，プログラムのソースコード全体の中で，特別なエンティティです．宣言部において指定される範囲といった属性を持ちます．

副型は，Ada の型付けモデルにおいて，まさに中心的な役割を果たします．また，Ada の型への特別な指示になります．

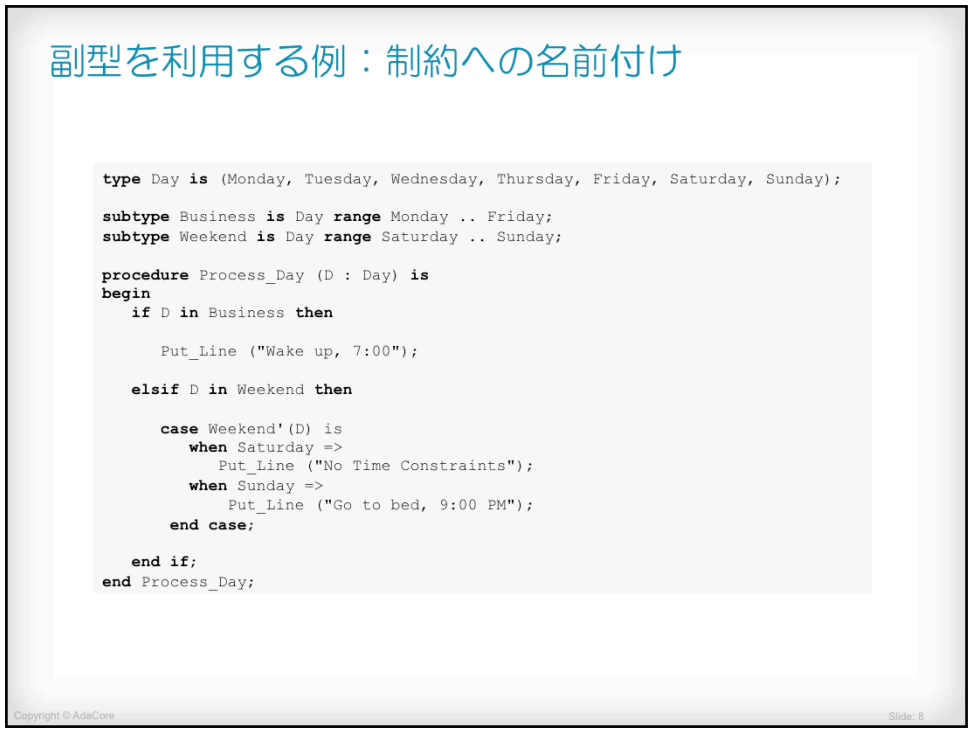
副型が宣言されたとき，付加的な制約を持ちますが，新規の型ではありません．実際，ある型とその副型の間では，追加の検査はありますが，操作が可能です．

上記のことを踏まえて，整数型のうちゼロ以上を指す Natural 副型の例を見てみましょう．最大の値は，Integer 型と同じです．2つめの副型は，1 以上の整数を指す整数型の副型である Positive です．これも，最大の値は，Integer 型と同じです．

V1 は，整数型で，Zero で初期化されています．

次に V1 を Natural 副型オブジェクト V2 に代入することは正当です．なぜならば，整数型 Zero は，Natural 副型の範囲内だからです．

しかし，コードの最後の行は正しくありません．なぜならば，V1 の V3 への代入は，Zero が，Positive 副型の範囲外であるため不正になります．



このコード断片の例で，新しい型 Day を扱います．Monday から Sunday までを列挙した列挙型です．

Day 型の 2 つの副型を宣言しています．一つは Business 副型で，Monday から Friday までの数え上げ制約を持っています．またWeekend 副型は，Saturday と Sunday の列挙を Day 型に対して制約として加えています．

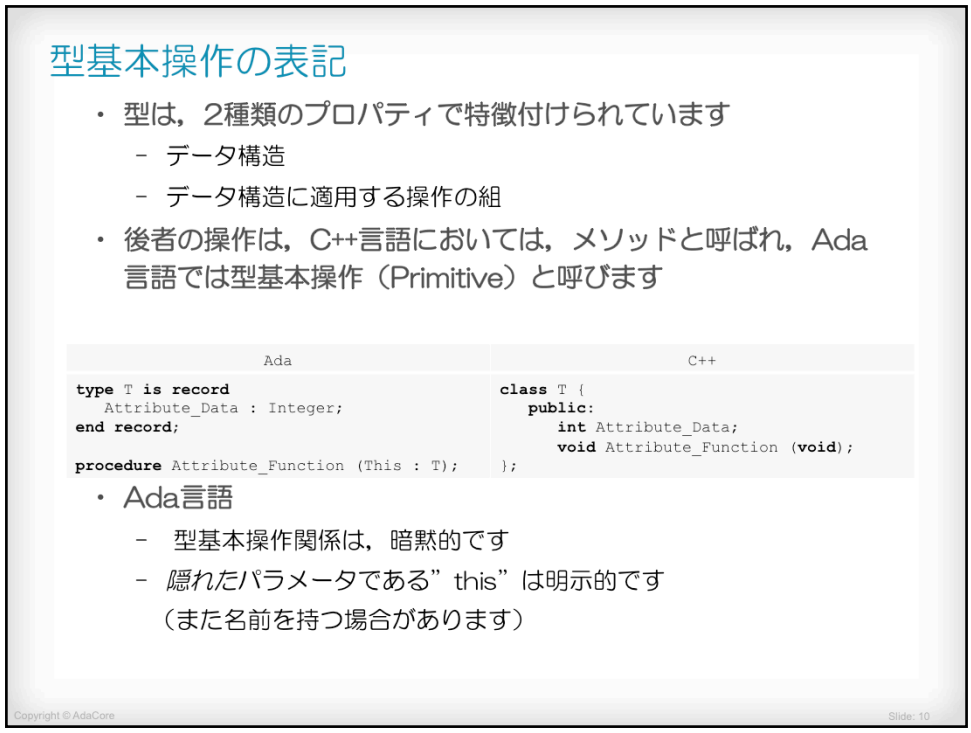
これは，とても分かりやすいはずです．次に，Process_Day と呼ぶ手続きがあります．これは D という Day 型のパラメータを受け取ります．

手続き Process_Day の最初の行は，そのオブジェクト D が，Buisiness 副型の範囲にあるかどうかを検査します．もし範囲内ならば，"Wake up, 7:00（午前 7 時に起きる）" と文字列を印字します．

逆に，副型が，Weekend の範囲であれば，明示的に D を Weekend 副型にキャストして，ケース文を使います．

これは，型と副型が，同じ型のように扱えることを示しています．しかし，通常，許される値に関する異なるビューを持っています．

このセクションは，型基本操作と，Ada の型が，どのように関係しているかについての説明です．

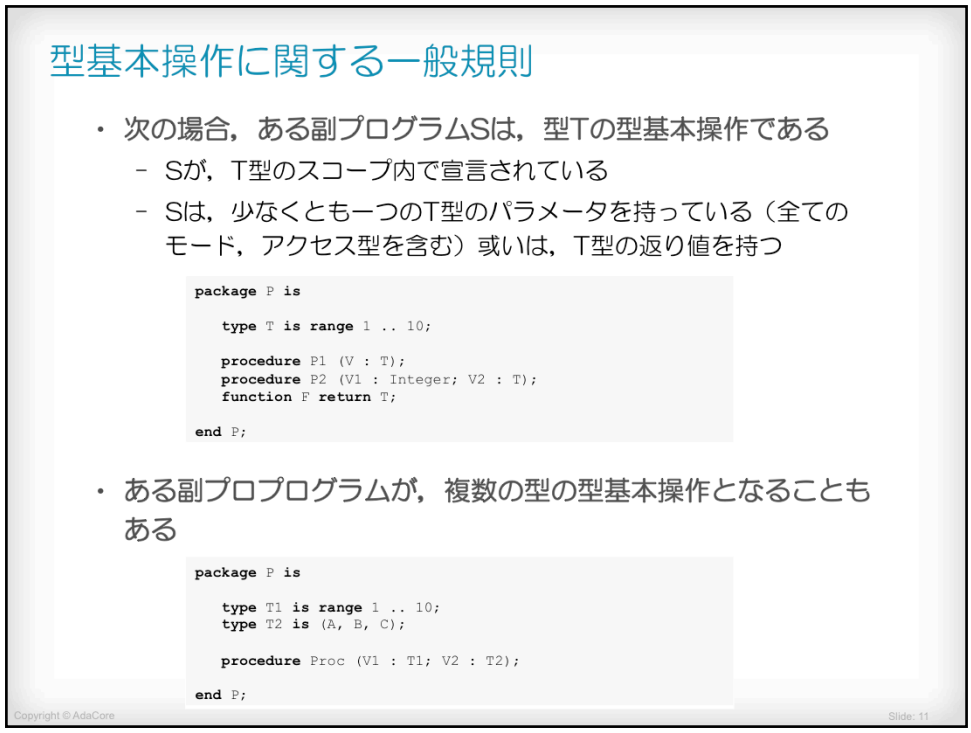


ある Ada の型を宣言するとき，2 種類のプロパティを持つエンティティとして示されます．一つがデータ構造であり，もう一つが，そのデータ構造に適用される操作の組です．

これはAda言語に特有というわけではありません．C++ 言語や Java のような他の言語において，これら操作をメソッドと呼びます．しかし，Ada 言語では，型基本操作（Primitive）という用語を用います．

ここに意味的に似たコードを Ada 言語と C++ 言語で示します．レコード型 T に対する Ada の型基本操作は Attribute_Function です．C++クラス定義では，パブリック関数として同じ名前を使っています．

Ada言語では，型と型基本操作の関係は暗黙的です．しかし，C++ でいう this，Pythonでいう self パラメータと同等のものを必要とします．しかし，Ada言語は正規のパラメータ名を持つことができます．



型基本操作を如何に宣言するかに関して，一般規則があります．それはどのスコープ含まれるか，パラメータの型は何かということです．

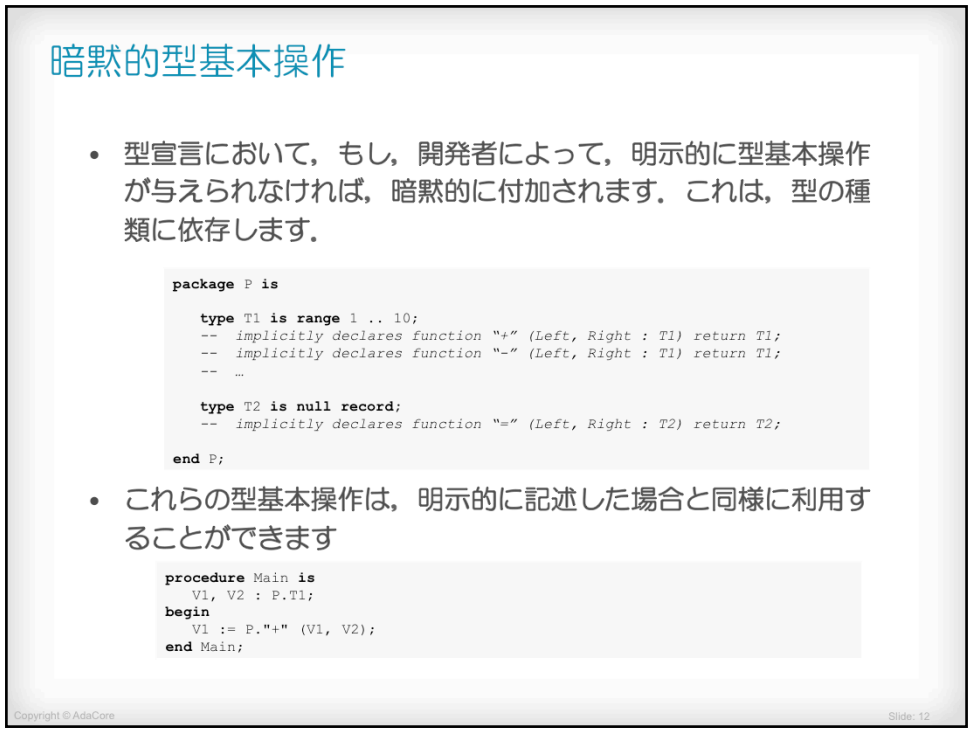
もし，操作 S を型 T の型基本操作として宣言したいのであれば，S の宣言は，T と同じスコープ内でなければなりません．

S は，モードによらず少なくとも型 T の一つのパラメータ（アクセス型を含む）を持つか，型 T の返り値を持たなくてはなりません．

例では， T 型 と 3 つの型基本操作を示しています．2 つの手続きがあり，T 型のパラメータの順序は重要ではないことを示しています．関数は，T 型の返り値を持つので，型基本操作になります．

上記の規則に従っていれば，操作が複数の型の型基本操作となることができます．

最後の例では，Proc は，T1 型と T2 型の型基本操作となります．スコープ規則とパラメータ型規則を満足しているからです．



型の種類に応じて，コンパイラは，暗黙的に型基本操作を作ることができます．

いま，範囲が 1 から 10 の数値型 T1 があります．この型に対して，コンパイラは少なくとも「加算」と「減算」の型基本操作を作ります．

レコード型 T2 の場合も同様です．暗黙的な等値判定操作が宣言されます．

これらの型基本操作は，開発者が宣言した型基本操作と同様に，完全なパッケージ修飾記法を用いて，利用することができます．

しばしば use 節はコーディング規則によって禁止されています．禁止することで可読性を向上を図ります．

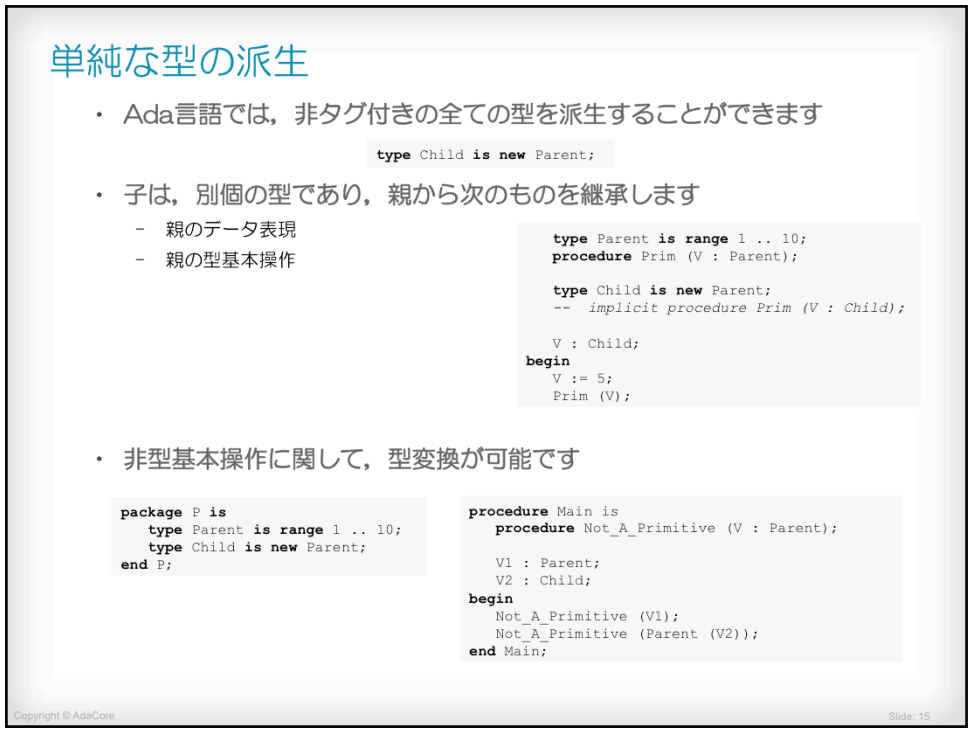
use 節を使わないと，パッケージ名を付けた完全な修飾付き操作名を使う必要があり，逆に，保守できないコードになったり可読性を減少することがあります．

この例が，最初のコード断片にある手続き Main です．くどくどと書かれた型基本操作は，入れ子のパッケージによって，さらに悪化しています．

use 節に類似した代替策があります．これは指定した型の型基本操作のみに可視性を与えます．

手続き Main の二番目の実現は，ずっと読みやすく，同一の意味となっています．

最後のセクションは，Ada 言語のもっとも強力な特徴の一つを扱います．それは，派生型を宣言する能力です．



新しい型を派生するためには，new キーワードと（次に）非タグ付き型の名前を書くことによって実現できます．

最初の例では，既存 Parent 型から，新しい Child 派生型を作っています．

派生型は，親の型が持つ既存の属性であるデータ表現と型基本操作を引き継ぐことができます

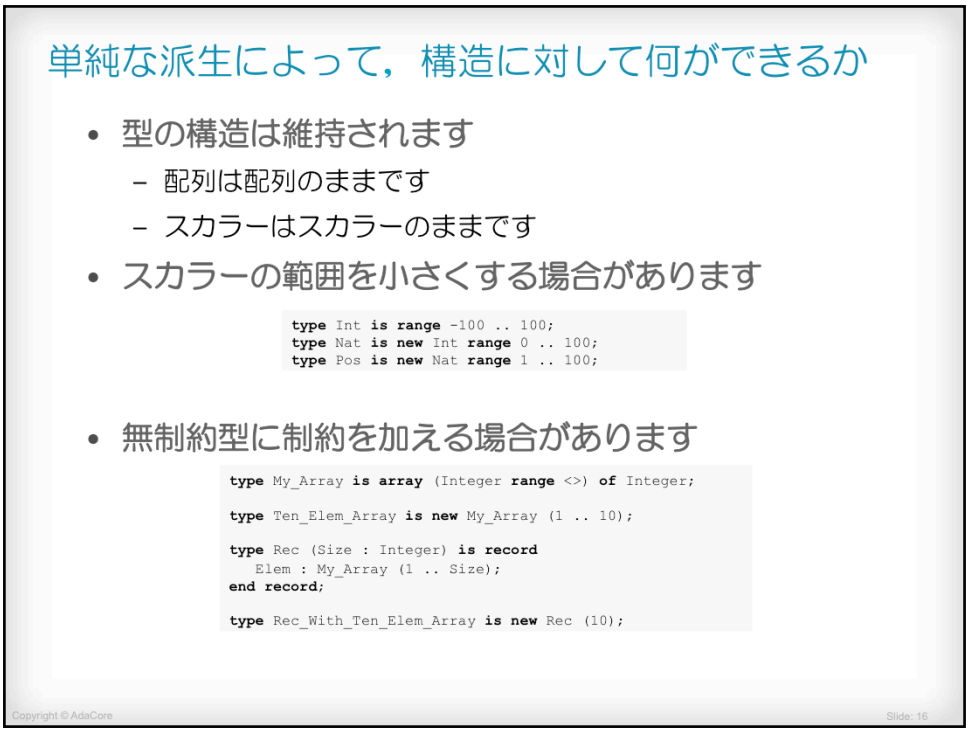
この例にあるコードの断片には，Parent という数値型があります．Parent は，Prim と呼ぶ型基本操作を持っています．この操作は，Child 型に先行して宣言され，Child 型は，Parent 型から派生しています．ボディ部において，Child に型づけられているオブジェクトを，Parent 型の型基本操作に対して，型変換を行うことなしに受け渡すことができます．子型は，型基本操作を親から継承しているため，可能になっています．

継承による利点はありますが，新しい型は完全に別個の型であり，型基本操作ではない操作に関しては，型変換が必要になるということは覚えておく必要があります．

最後のパッケージ P の例を見ます．ここでは，範囲が 1 から 10 の数値型 Parent を宣言しています．次に，この Parent 型から新しい子となる Child 型を派生します．Child 型は Parent 型から幾つかの利点を継承しますが，全くの別個の型です．

次に，Main 手続きを宣言します．この手続き中で，Not_A_Primitive 操作を宣言します．なぜ，これは型基本操作とならないのでしょうか．この操作は，型宣言と同一のスコープ内にはありません．従って可視性規則によって，型基本操作とはなりません．

手続き Main 中では，次にParent 型，Child 型の2つのスタック領域オブジェクトを宣言しています．手続きのボディ部では，Not_A_Primitive を呼び出しますが，一つは通常通り，Parent



これまで見てきたように，派生型は，派生元の型属性を継承することができます．

派生型を宣言するときに，継承する属性に対して，（制限はありますが）修正を行うことができます．

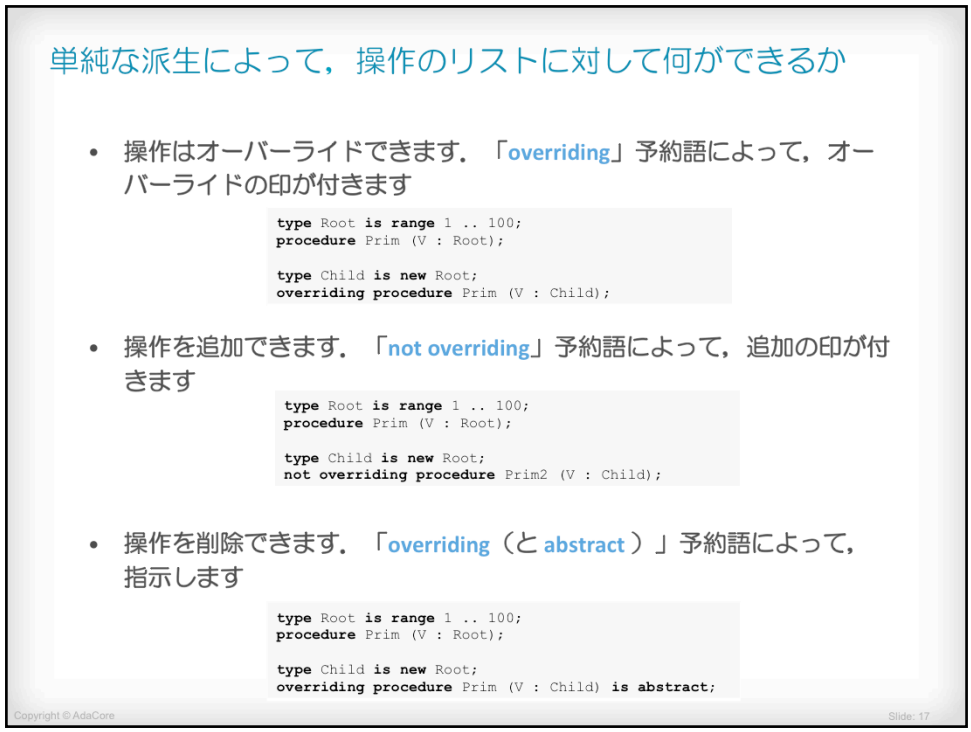
幾つかの制限があります．例えば，配列をスカラーに変更することはできません．逆もできません．基本的な構造は維持する必要があるからです．

最初の例で見るように，派生によって，スカラー型範囲を小さくすることができます．

Int 型は，ここでは- 100 から 100 までの範囲を持っています．派生した型，Nat と Pos では，その宣言中で継承した範囲を小さくしています．

無制約型は，派生型の宣言で範囲制限を追加できます．これは，強力な特徴です．

最後の例に，My_Array という無制約配列型があります．Ten_Elem_Array を派生するときに制約を加えています．レコード Rec の要素の範囲についても同様に制限を加えています



データ構造属性は，親の型から継承され，派生時に操作することができます．しかし，型基本操作も同様に操作できます．

次の点から始めます．継承した型基本操作を，継承型自身が，同名の型基本操作で置き換えることが可能です．「overriding」を派生する型の型基本操作の前に付けて宣言することで可能となります．

ここで， Root は，Prim という型基本操作を持っています．派生した子型は，この型基本操作を自身のバージョンで置き換えています．

派生型は，派生型自身の型基本操作を付け加えることもできます．「not overriding」 を前につけることで可能になります．

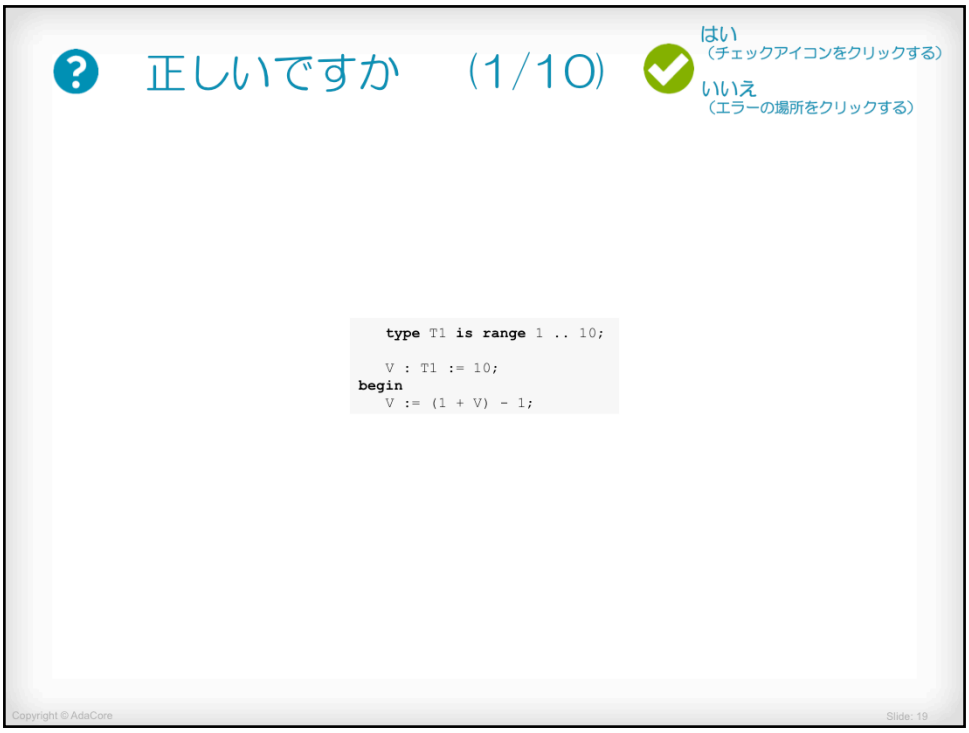
二番目のこの例では，Prim2 は，Child の型基本操作であって，Root の型基本操作ではありません．

親から型基本操作を継承しないという派生型を作ることもまた可能です．これは「overriding」を前につけ，その操作が抽象である(「is abstract」）ことを示すことによって表現できます．

いよいよこの講義の最後のセクションとなりました.

Ada の型付けに関して十分な知識を持ったことでしょう. あなたの理解度を確認するちょっとした問題をやってみましょう.

幸運を！



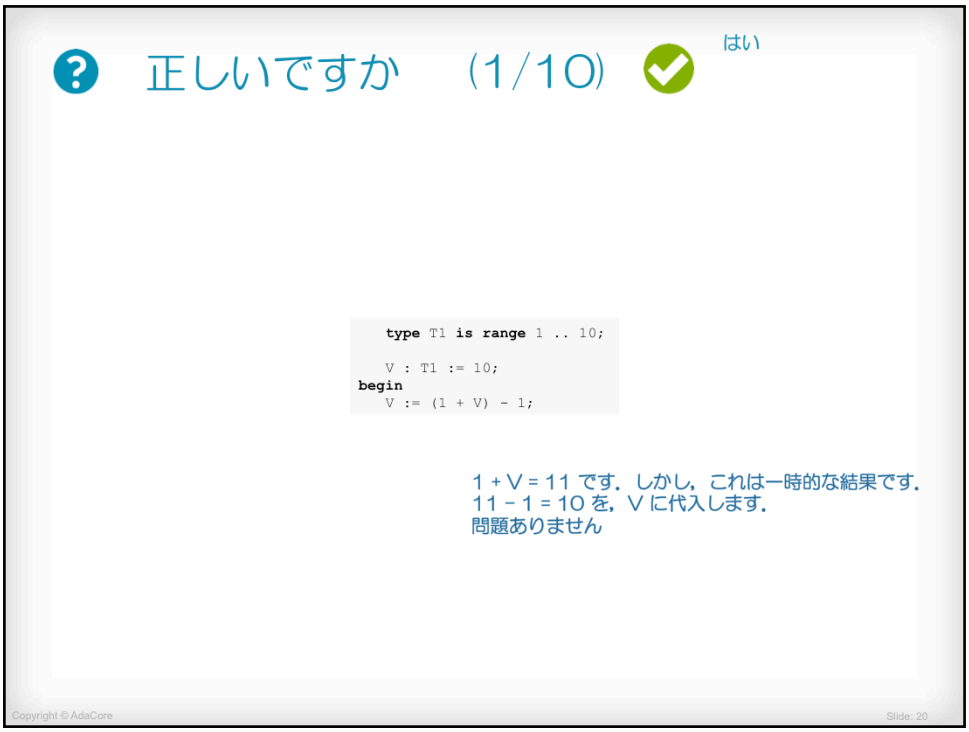
最初の問題では，数値型 T1 を導入します．範囲は 1 から 10 です

実体 V を作り，明示的に10 で初期化します．

ボディ部では，V に (1 + V) – 1　を代入しています．

V への代入は正しいでしょうか？

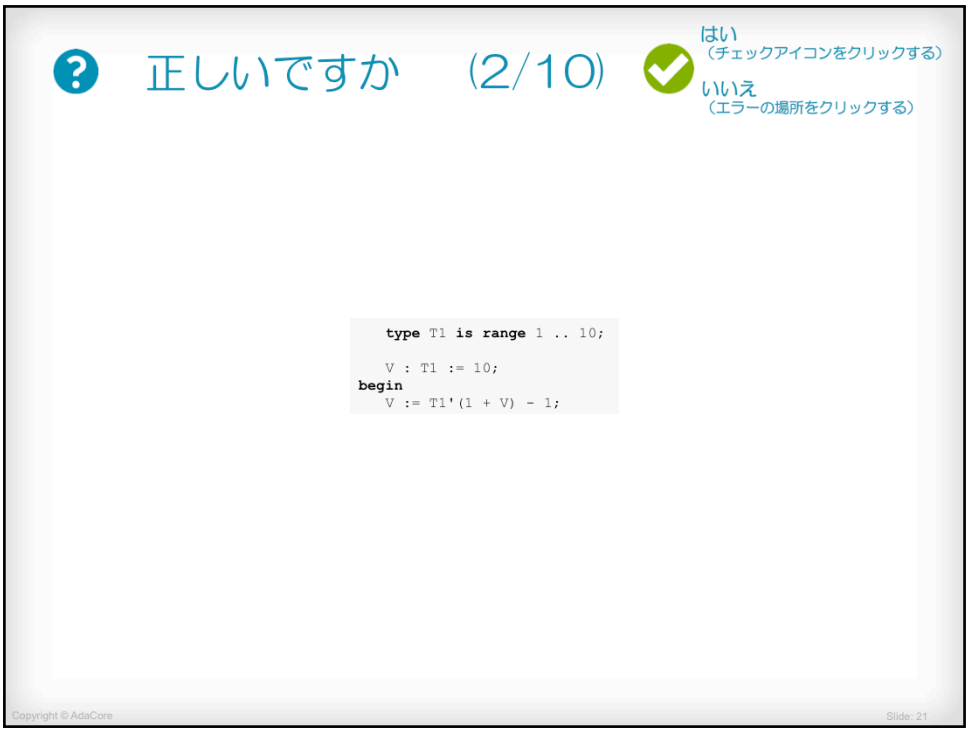
このコードが正しいと思うときは，「はい」のアイコンをクリックしてください．間違いと思えば，その場所をクリックしてください．



コードは正しいです．

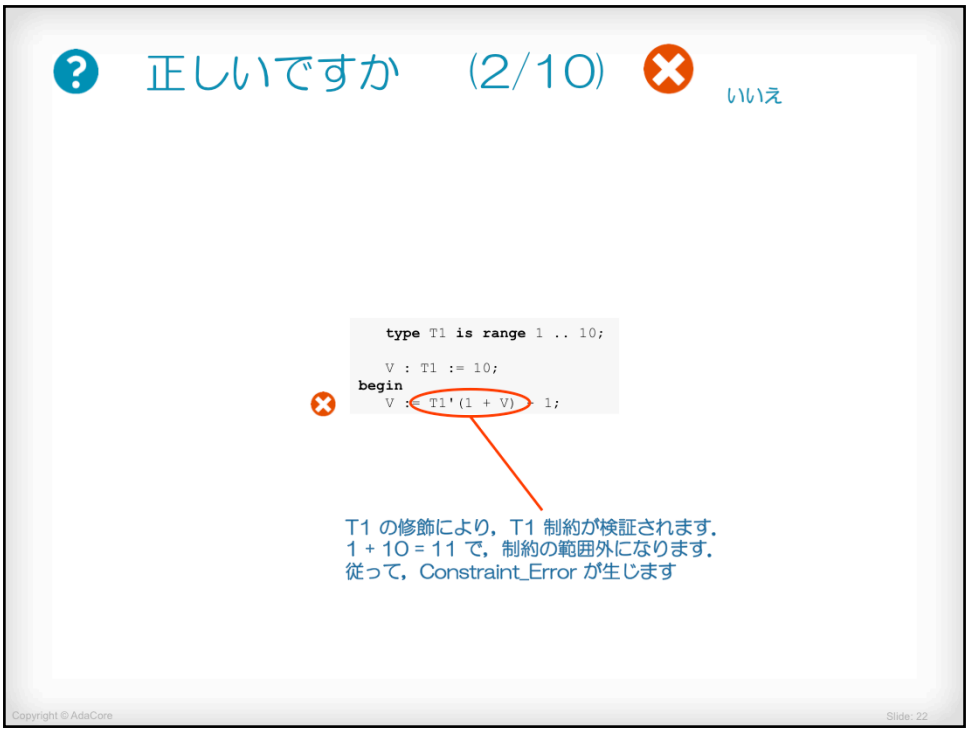
1 + V は 11 となります．しかし，これは一時的な値であり，副式においては，範囲検査が行われないことを既にご存じと思います．代入が実際に生じるときには，11 の値は 1 を減じるので，T1 型の範囲に収まります．

問題はありません．



二番目の問題は，前回の問題を元にしています．しかし今回，副式を型 T1 によって修飾しています．

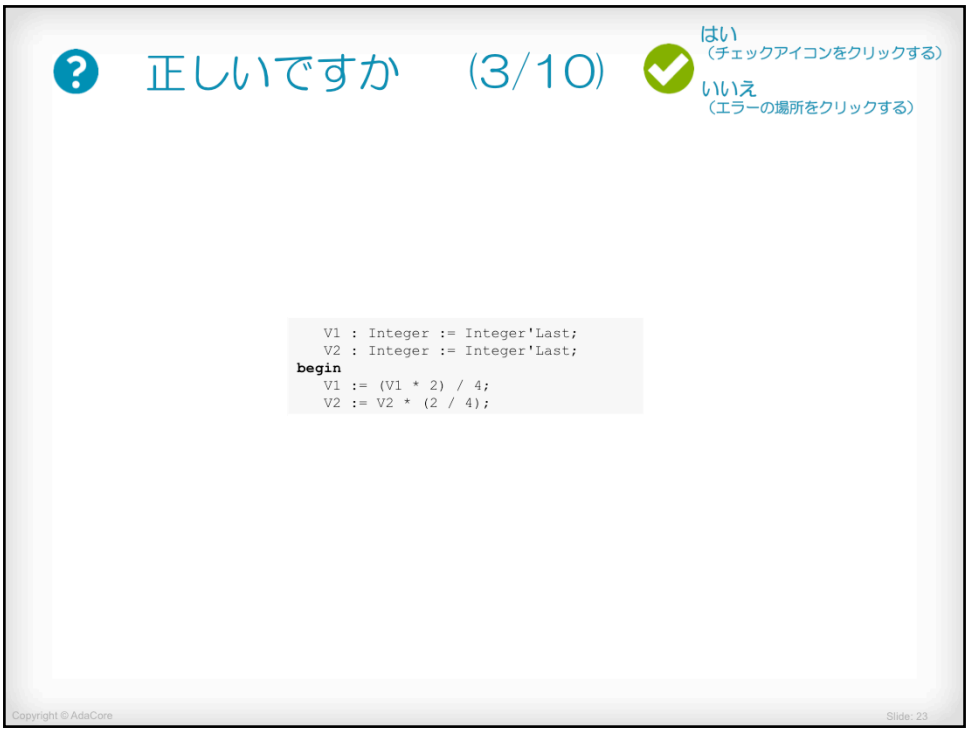
このコードが正しいと思うときは，「はい」のアイコンをクリックしてください．間違いと思えば，その場所をクリックしてください．



これは正しくありません．

T1 の修飾により，副式でオブジェクトの代入が生じます．従って範囲検査が行われます．

1 + V は，T1 の許された範囲を超えます．従って， CONSTRINT_ERROR が生じます．

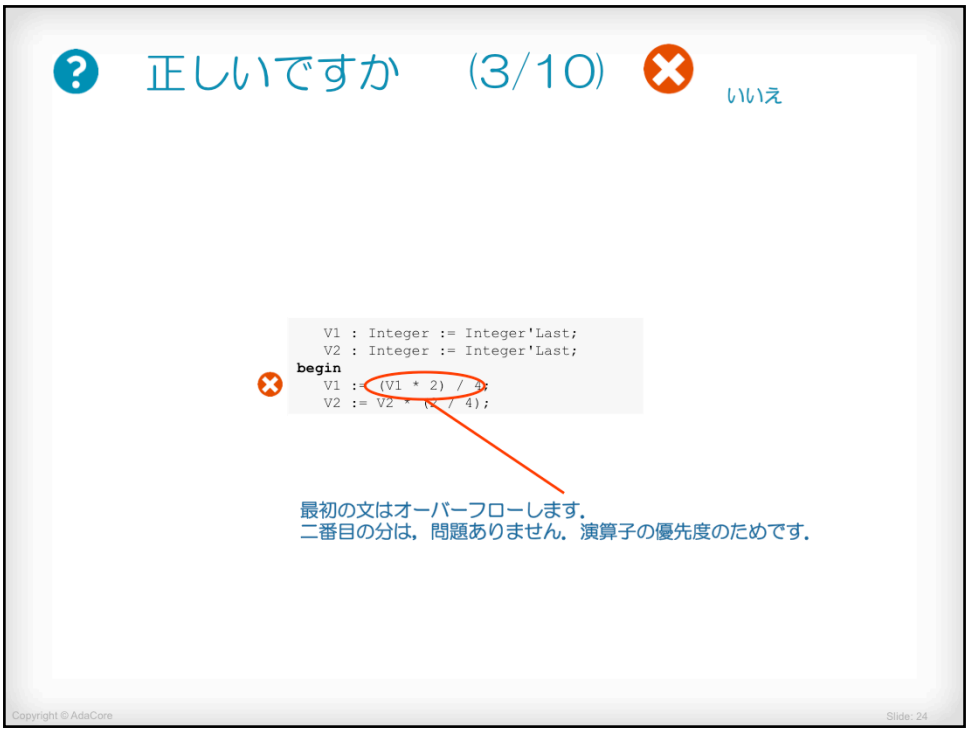


次の問題では，2 つのスタック領域オブジェクトを扱います．整数型であり，明示的に，整数型の最大値で初期化しています．

ボディ部には，二つの文があります．最初は，V1 を 2 倍にして，4 で割ったのちに，V1 に代入しています．

二番目の文では，2 を 4 で割った結果を用いて V2 との積を求め，V2 に代入しています．

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．



コードは正しくありません

最初の文は，印を付けた箇所でオーバーフローし，CONSTRAINT_ERROR 例外を発生します．

二番目の文は正しい．演算子の優先度とかっこを使用しているためです．

4 番目の問題では T 型を宣言しています．これは Integer 型から派生しており，継承した範囲に制約があります．

2 つのスタック領域オブジェクトを作っています．整数型の V1 は，明示的に初期化されています．派生型である T 型の V2 は，明示的に V1 を用いて初期化しています．

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．

このコードは誤っています．

T 型と整数型の 2 つは異なった型です．直接代入することは不正です．

派生型は，データ構造と型基本操作以外は，何も親の型から継承しないことを，思い出して下さい．

次の問題は，先の問題を少しだけ変更しています．

どこを変更したかは述べません．ご自身で探してみて下さい．

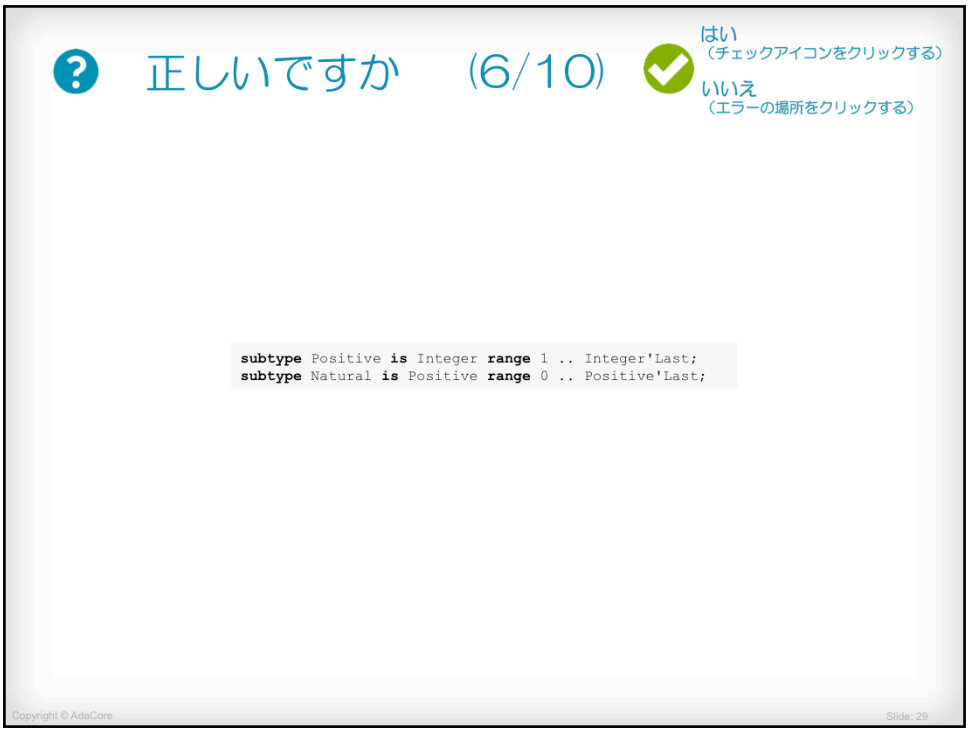
このコードが正しいと思うときは，「はい」のアイコンをクリックしてください．間違いと思えば，その場所をクリックしてください．

このコードは間違っています．

T 型は，ここでは整数型から派生したものではなく，整数型の副型となっていることに気づかれたことと思います．

このことは，T 型と整数型が同じ型であることを示しています．従って，互いに代入することは，完全に正しいこととなります．

しかし，V2 への代入は，V1が 1 と等しいか，それより大きいことを必要とします．ここでは，そうなっていないので，CONSTRAINT_ERROR 例外が送出されます．



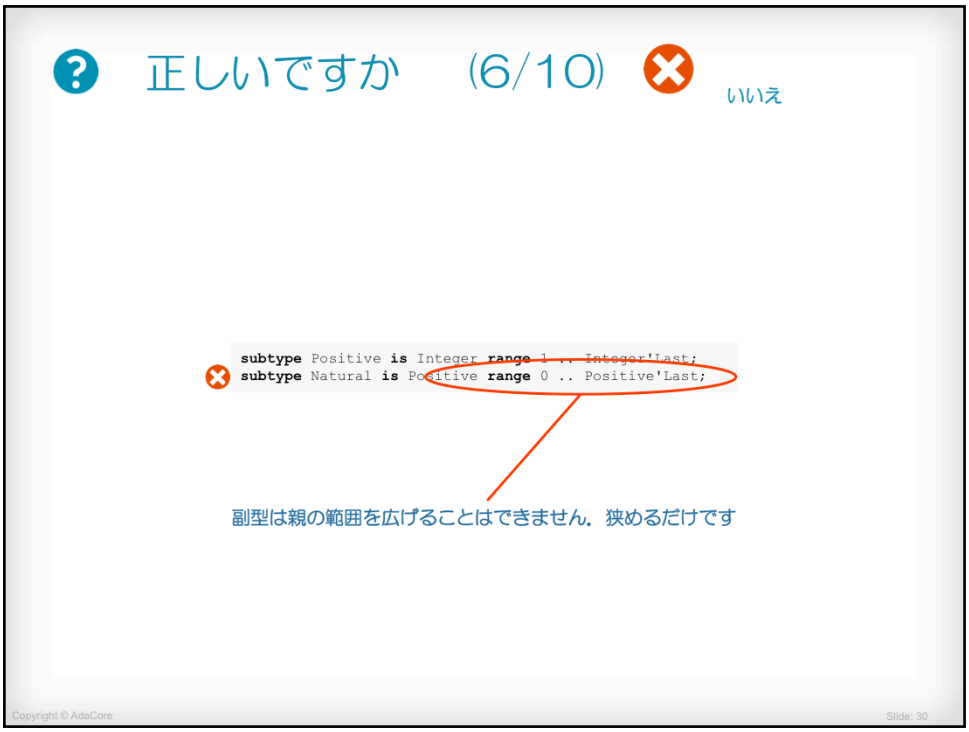
6 番目の問題は，より複雑な例を通じて，副型の利用に焦点を当てます．

2 つの副型があります．Positive と Natural です．

Positive は，整数型の副型であり，範囲制限は，1 から，整数に許される最大値です．

Natural 型は，Positive 型の副型であり，その範囲は，0 から Positive に許される最大値です．

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．

コンパイルできますが，CONSTRAINT_ERROR 例外が実行時に送出されます．

問題は，副型 Natural の範囲にあります．そこには 0 が含まれますが，親の型は，1 から始まっています．

副型は，親の範囲を広げることはできません．しかし，狭めることはできます．

次の問題は，型基本操作の規則に対する理解を確認するためのものです．

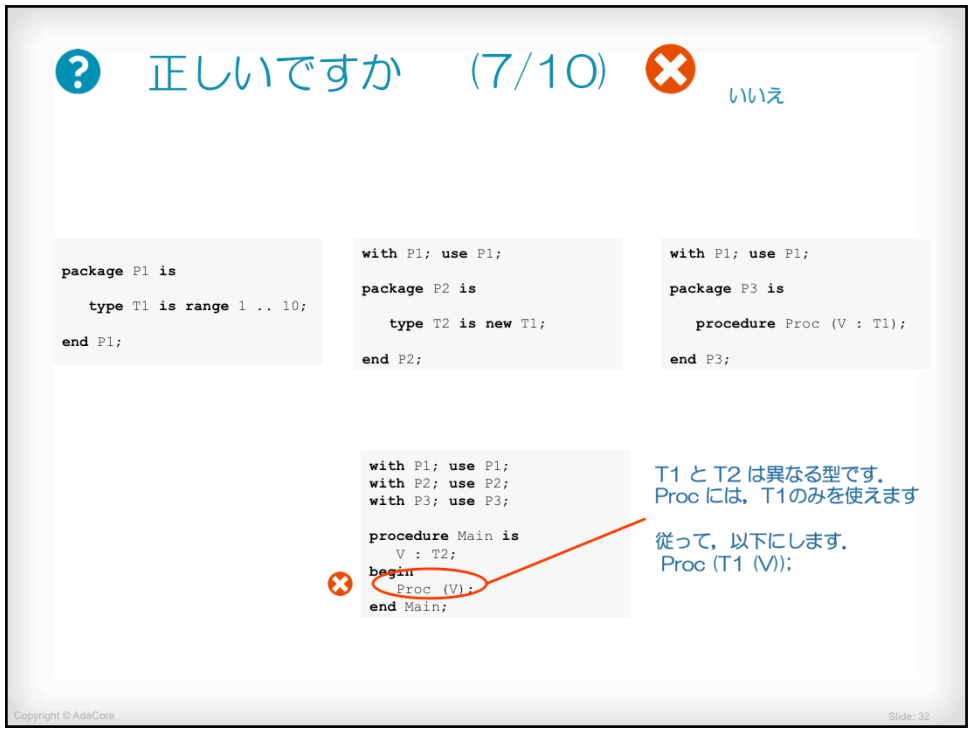
パッケージ P1 は，1 から 10 の範囲のスカラー型を宣言しています．パッケージ P2 は，T1 から派生した T2 型を持ちます．

最後のパッケージ P3 では，操作を宣言しています．その操作は，パッケージ P1 の T1 型の単一のパラメータを持ちます．

パッケージ Main では，3 つのパッケージを全てスコープに入れます．ここではスタック領域オブジェクト V を，パッケージ P2 の T2 型として宣言しています．

次に，スタック領域オブジェクト V をパラメータとし，パッケージ P3 にある操作 Proc を呼び出しています．

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.

手続き Proc は，T1 の型基本操作であり，T2 の型基本操作ではありません．明示的な型変換を行うことなしに，T2 型のオブジェクトを渡すことはできません.

次の問題では，パッケージ P1 内部で，T1 型に対する Proc という型基本操作を導入しています．

パッケージ P2 は，T2 型を宣言しており，これは，パッケージ P1 の　T1 型の派生型です．

Main 手続きは，型 T2 のスタック領域オブジェクトを作り，Proc 呼び出しにおいて，パラメータとして渡します．

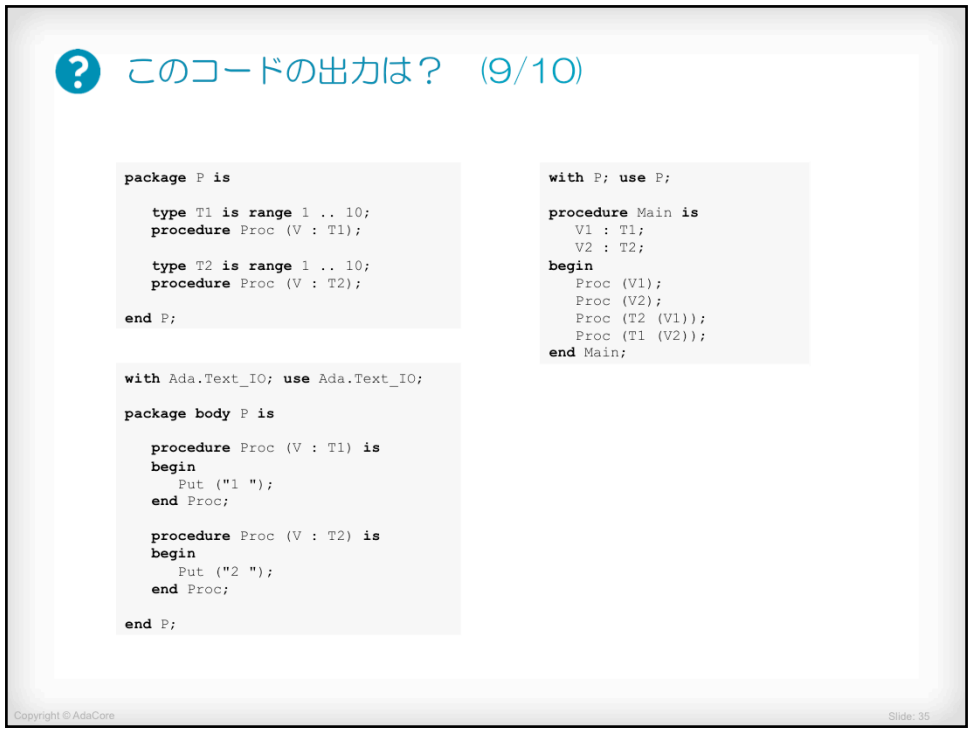
このコードが正しいと思うときは，「はい」のアイコンをクリックしてください．間違いと思えば，その場所をクリックしてください．

これは正しいコードです．

T2 は派生型であり，派生型は，データ構造と全ての型基本操作を親の型から継承します．

この例における親の型は T1 であり，Proc と呼ぶ型基本操作を持っています．

従って，Proc は継承されているので，型 T2 のオブジェクトを直接に渡すことができます．



9 番目の問題は，型基本操作に関する知識を確認するためのもう少し複雑な例です．

パッケージ P は二つのスカラー型を宣言しています．それぞれ，Proc という名前の型基本操作を持っています．

パッケージボディ部は，型基本操作の実現を持ちます．それぞれ異なった値を出力します．

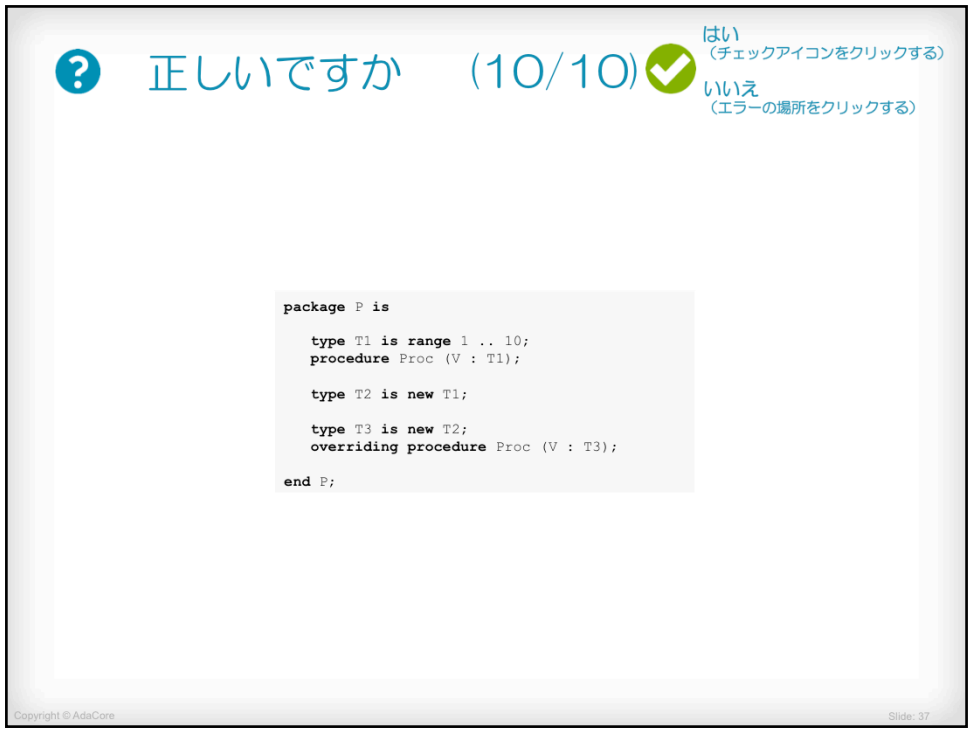
手続き Main では，2 つの型のスタック領域オブジェクトを宣言しています．次に Proc を何度か呼び出します．このとき，直接オブジェクトを渡す場合と型変換して渡す場合があります．

どういうテキストが出力されるでしょうか．

もし答えが分かったら，次のページを見て下さい．

テキスト出力は，1 2 2 1 となります．

型変換により，型変換後の最終的な型に対して，正しい型基本操作を呼び出すことができます．

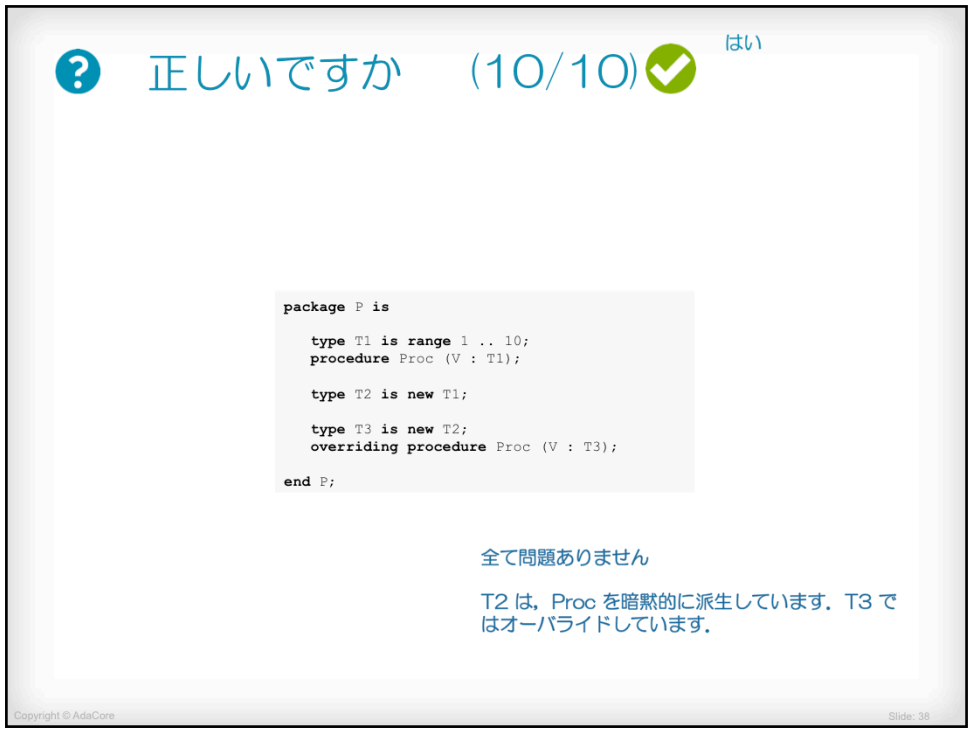


最後の問題は，派生型階層に関するものです．T1 型が元となります．この型は，Proc という名の型基本操作を持ちます．

T2 型は，T1 型から派生しています．

T3 型は，T2 型から派生しています．継承した型基本操作 Proc を自身の実現に置き換えています．

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．



最後に来ました．このコードは正しい．

T2 型は，T1 の派生型なので，Proc を暗黙的に継承しています．T3 は，更に，Proc を継承し，明示的にオーバーライドしています．

Ada 大規模プログラムの Ada 型付けの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.