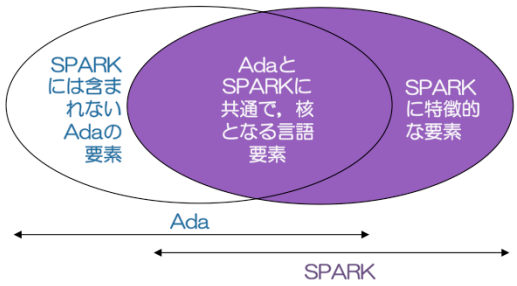

AdaCore ユニバーシティがお送りする　SPARK 2014 の最初の講義です．

この講義では，最初に SPARK2014 が提供する技術とツールセットの概要を示します．次に，幾つかのクイズを通して，理解度を確認します．

## SPARK 2014 – それは何か？

- プログラム言語です
  - Ada言語のサブセットです．静的検証を目的としています
  - プログラム仕様を向上させるための特徴を持っています




- プログラム検証ツール群です

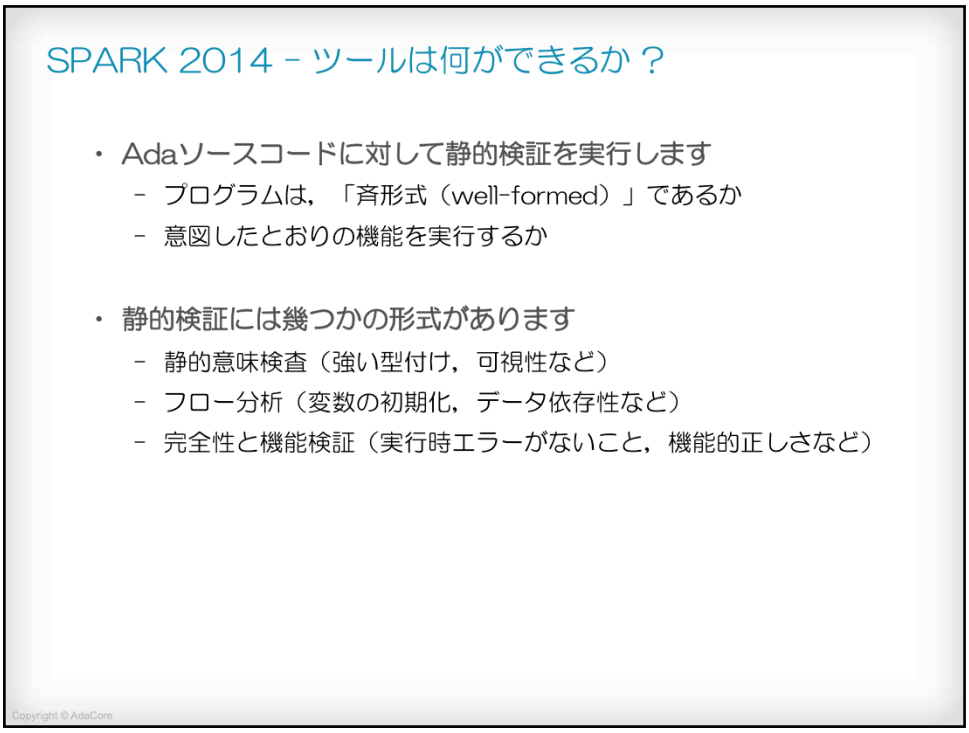


SPARK 2014 では，次の 2 つの役割を重視しています．一つは機能仕様と静的検証を行うプログラム言語としての役割．もう一つは，開発・検証ツールとしての役割です．

SPARK2014 言語は，Ada 言語のサブセットを元としています．安全性・信頼性が重要なソフトウェア開発で利用されることを目的としてきました．従って，形式検証に特に適しています．

Ada 2012 では，サブプログラム「契約」に用いるアスペクトを導入しています．SPARK 2014では，静的検証を支援するために，独自のアスペクトを加えています．

【訳注】　ここでの「契約」は，「契約による設計（Design by Contract）」というときの，契約と同じです．呼び出し側が約束を守ってくれれば（事前条件），結果を出す（事後条件）というのが中心的なアイデアになります．




最初にプログラムの静的検証を行う必要があります．直接ソースコードの検査を行います．プログラムをコンパイルしたり実行する必要はありません．

この検証には静的解析のためのツールを使用します．一般的には，検証には幾つかの形式があります．

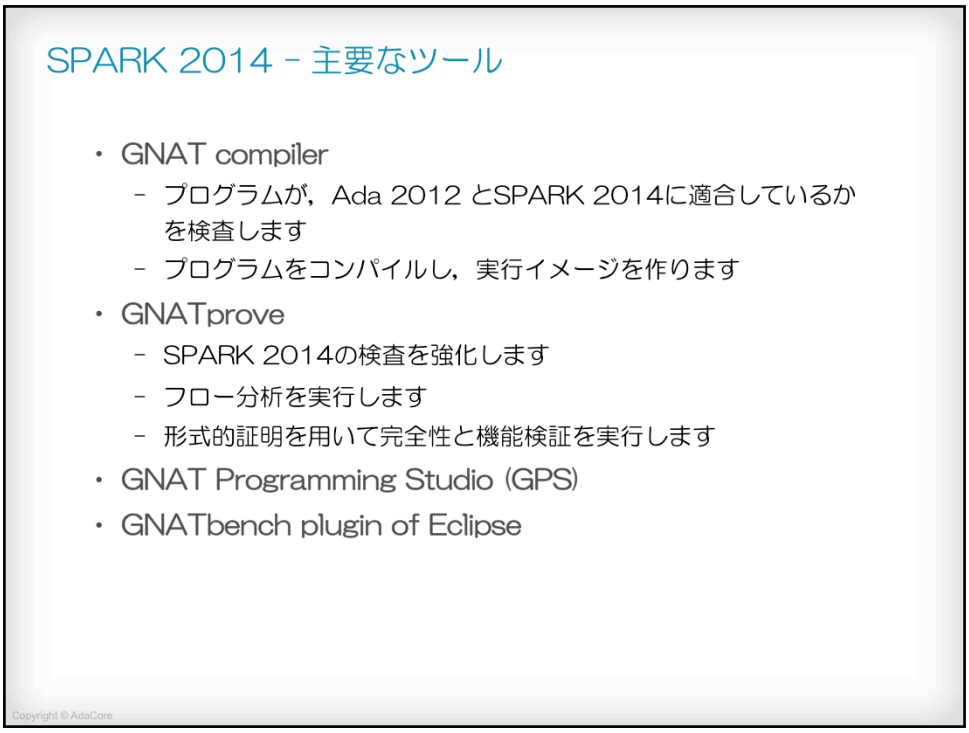
例えば，型検査，可視ルールの強制などです．もちろん，コンパイラはこれらの検査を行いますが，より複雑な推論に基づき実施します．それには，AdaCore社のツールであるCodePeer の持つ抽象的解釈が含まれます．

2つの異なる形式の静的解析を，ツールを用いて実行します．

最初は，フロー分析です．通常，もっとも高速に解析ができる形式です．特に，変数の初期化やサブプログラムの入出力データ間の依存性を検査します．

また，使用していない値の割当や，更新されていない変数を見つけることが可能です．

二番目は，形式的証明です．特に，実行時の誤りがないことを検査します．また，Ada 2012 の契約に適合しているかの検査を行います．

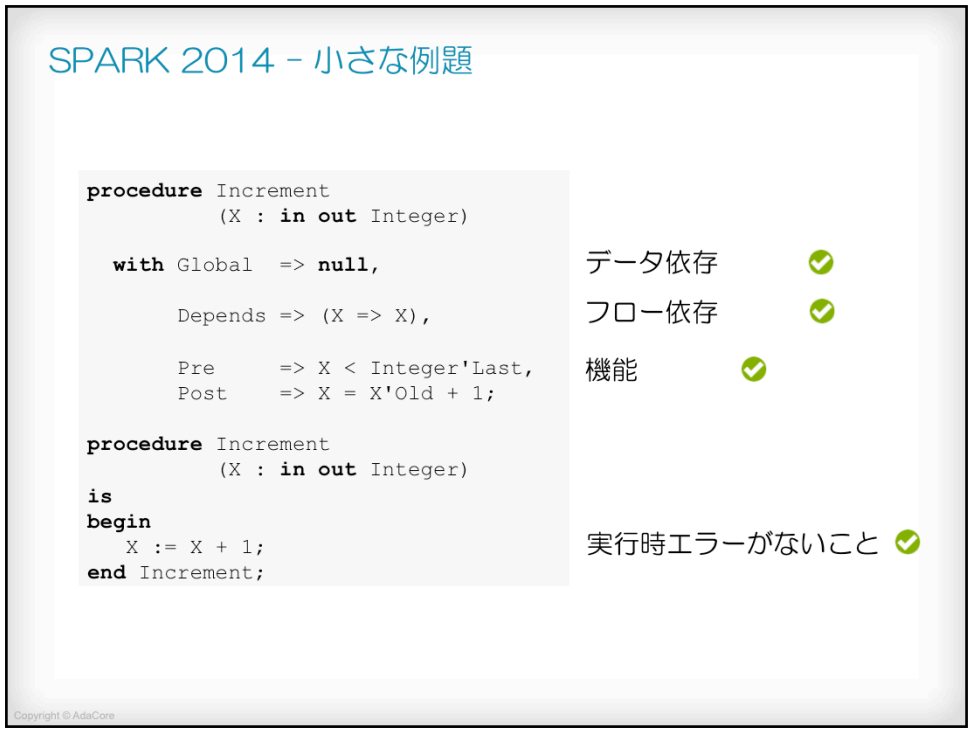



GNATproveは，SPARK 2014 言語に対応した形式検証ツールです．

SPARK サブセットに対する適合性検査を行います．またソースコードに対するフロー分析・形式検証を行います．

SPARK 2014 での開発を支援するための，他のツールもあります．

特に，GNAT コンパイラと GPS 開発環境は，SPARK 2014 での開発を十分に支援します．




Ada 2012 の簡単なサブプログラムの例を見てみます．このプログラムは，SPARK 2014 の特徴である検証可能なサブプログラムの「契約」について記載しています．

Increment サブプログラムは，パラメータ X に対して 1 を追加しています．

サブプログラム上には，契約概念を用いた幾つかのプロパティを記述することができます．

SPARK 2014 の広域変数（Global）アスペクトは，手続き Increment がいかなる広域変数の読み書きもしていないことを示しています．

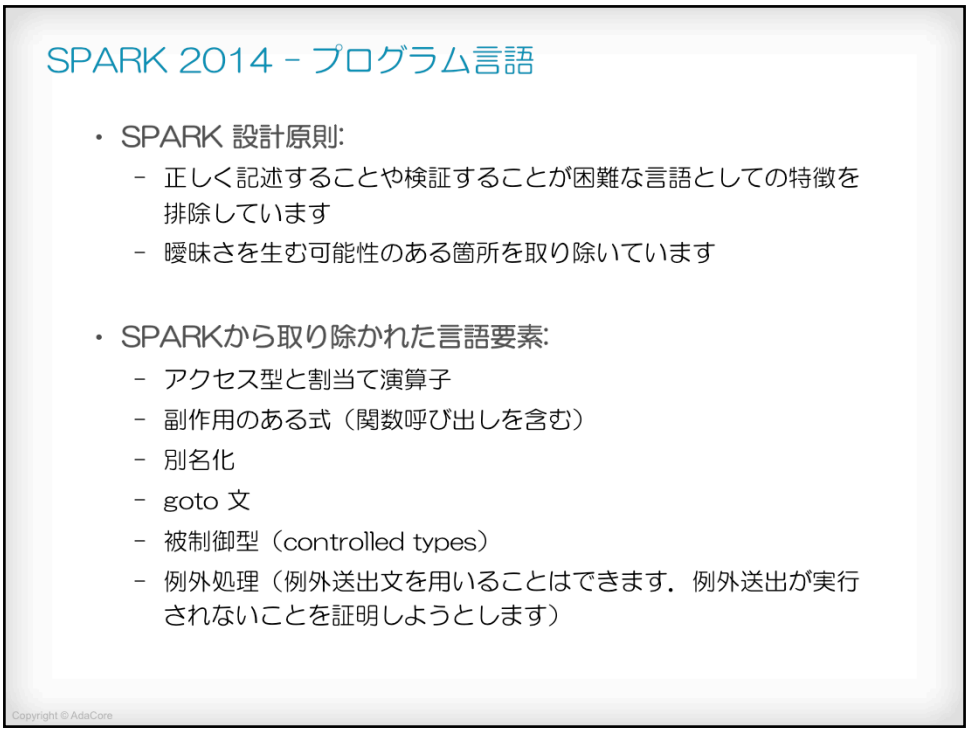
SPARK 2014 の依存（Depends）アスペクトは，サブプログラムのセキュリティという側面から興味深いものです．呼び出し後のパラメータ X の値は，呼び出し前の X の値にのみ依存していることを示しています．

Increment の機能的特性は，Ada 2012 の持つ事前（Pre）および事後条件(Post) アスペクトを用いて記述します．

Increment を，呼び出すことができるのは，X の値が Integer の最大値（Integer'Last）より小さいときのみです．

加算の操作は，サブプログラムのボディ部でオーバフローしないことを保証する必要があります．

最後に，Increment は，Xに対して加算の操作を確実に実行するということを記述します．呼び出し後の X の値は，呼び出し前の値より 1 大きくなければなりません．



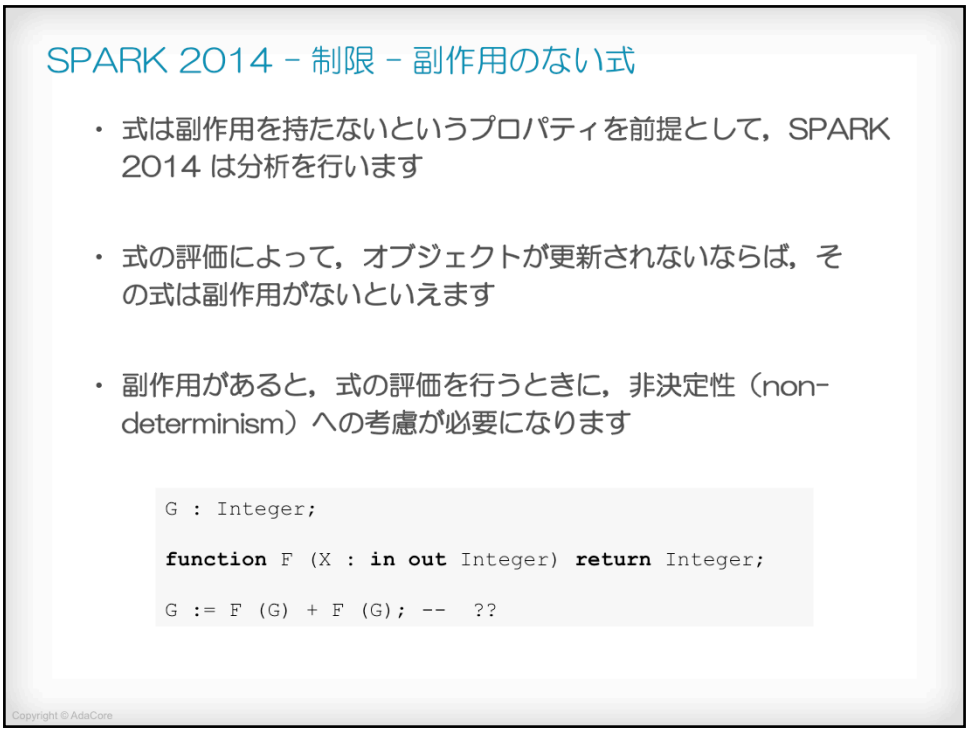

ここまでで，SPARK と Ada 言語の違いとその背後にある設計理由を理解することができたと思います．

Ada のサブセットとして，SPARK を設計したときの目標は，仕様を記述しやすく，しっかりした検証ができる，もっとも大きなサブセットを作ることでした．

除外したもののうち，最も特徴的な言語要素には次があります．アクセス型・割当て演算子と例外処理です．これらを実際に使う場合，ユーザ自身が多くの注記を加える必要があります．

goto文や被制御型（Controlled Types）は，制御フローに大きな影響を与えるため，サポートしていません．

残りの二つは，式の持つ副作用と，名前の別名化です．次から，これらについて詳しく見てみます．

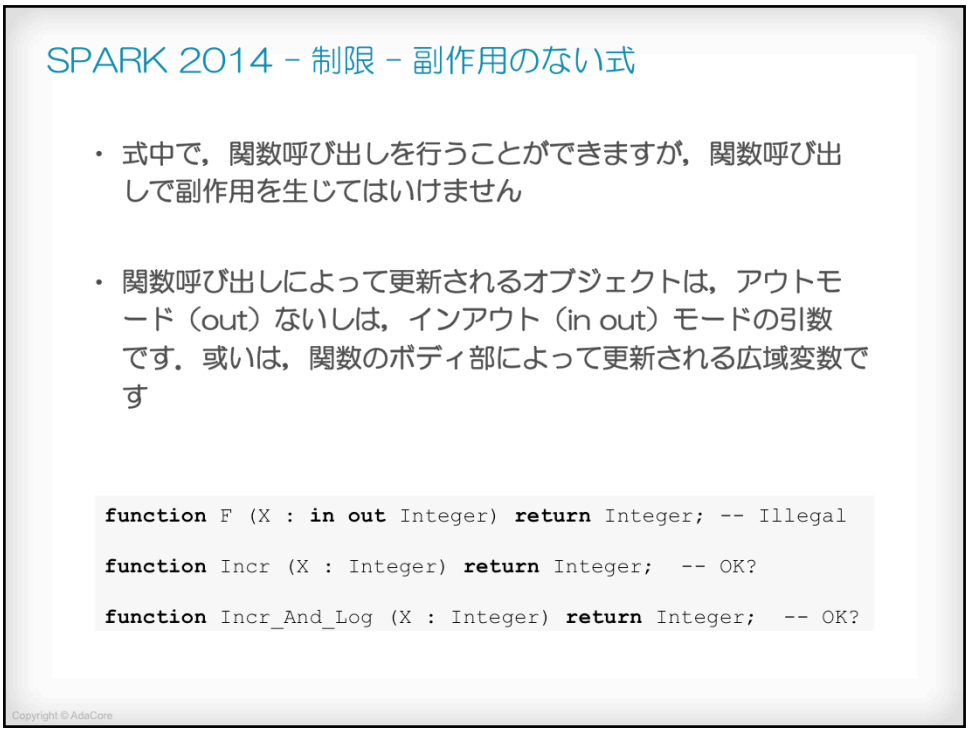



SPARK　言語では，式は副作用を生じません．即ち，SPARK では式の評価で，オブジェクトを更新することはできません．

この制限は，予測不能なふるまいを避けるために必要です．例えば，評価順や，パラメータの受け渡し機構，コンパイラの最適化で生じる問題を解決するために必要です．

下図の例では，２つの関数Fの評価順で，値が変化するという非決定性があります．従って，SPARK 2014では，不正な式となります．

（訳注）オブジェクトは，ここでは変数・定数のことを指す．80年代には，算体と訳されていました．




式の静的検証を支援する必要があります．関数呼び出しは，それ自身が式であるため，副作用を避けなければなりません．

関数が潜在的に副作用を生じる場合として，パラメータあるいは広域変数の更新があります．

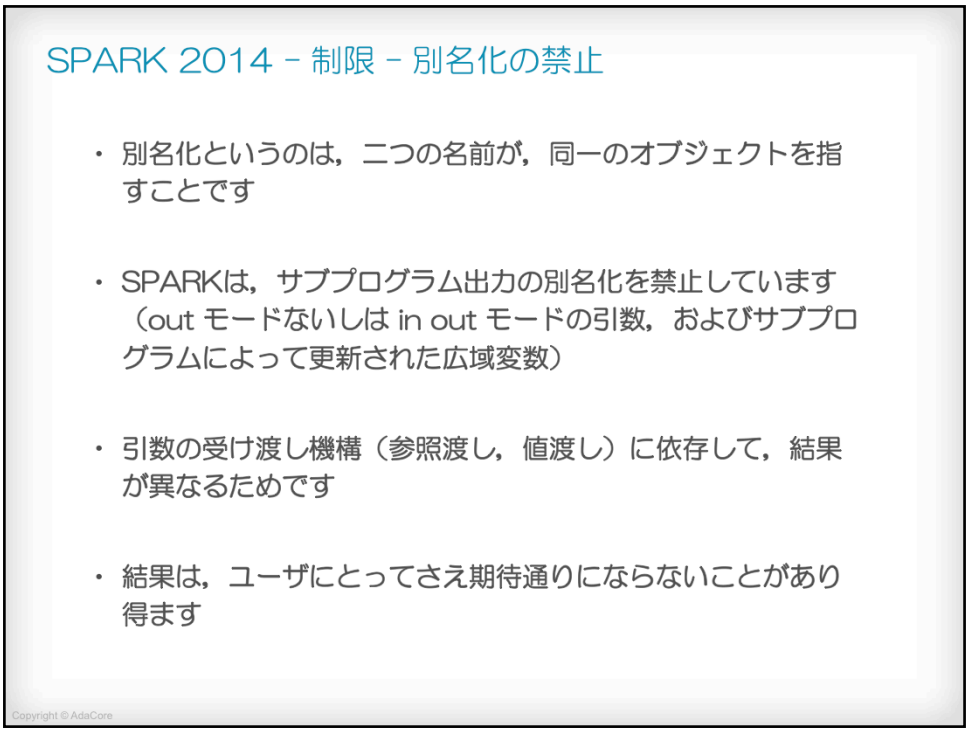
従って，SPARK 2014 では，out　ないしは in out 引数を持った関数であるサブプログラムを禁止しています．関数 F や，広域変数を用いた関数が相当しています．

多くの場合，これらの関数は，容易に手続き（procedure）によって書き換えることができます．


ボディ部にアクセスし，SPARK ツールは，関数が本当に副作用を持たないかを確認します．

ここに例を示します．関数 Incr と 関数 Incr_And_Log は，共に同じシグネチャを持ちます．

Incr は，正当な　SPARK コードですが，Incr_And_Log はそうではありません．
Call_Count 広域変数を更新しているからです．




SPARKにあるもう一つの制限は，別名化です．

2つの名前が，同一のオブジェクトを参照しているとき，別名化と呼びます．

アクセス型は，SPARKでは認められていません．従って，別名化は，手続き呼び出しにおいて，パラメータを受け渡すときにのみ生じる可能性があります．

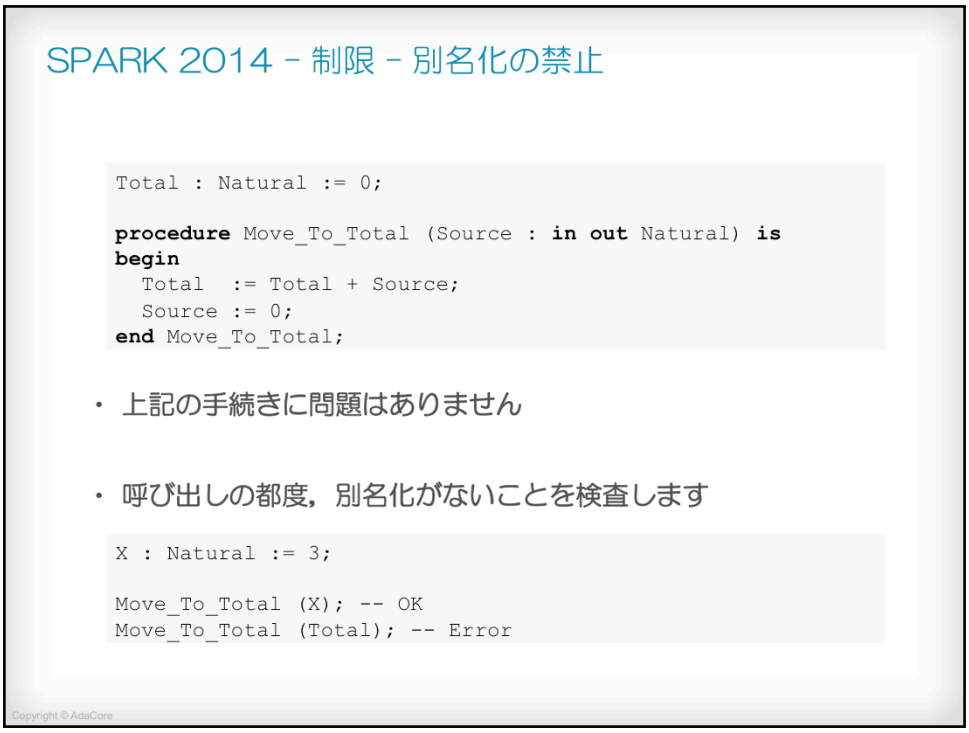
結果として，手続きが呼び出されたときに，out モードや in out モードの引数がないこと，或いは，手続き本体部で，広域変数を更新していないことを，SPARK　は確認します．

SPARK で，別名化を禁止する 2 つの理由があります．

第一の理由は，検証がより困難になるためです．異なる名前を持つ 2 つの変数を更新している記述が，実際には同一のオブジェクトを更新しているかもしれない，ということを考慮する必要がでてくるからです．

第二の理由として，ユーザにとっても，結果は期待はずれになるかもしれないからです．実際，引数が別名化されているとき，サブプログラム呼び出しの結果は，引数の受け渡し機構と同様に，コンパイラの処理に依存する可能性があります．

更に，プログラマが，別名化から生じる可能性を余り考慮しなかったということがあります．



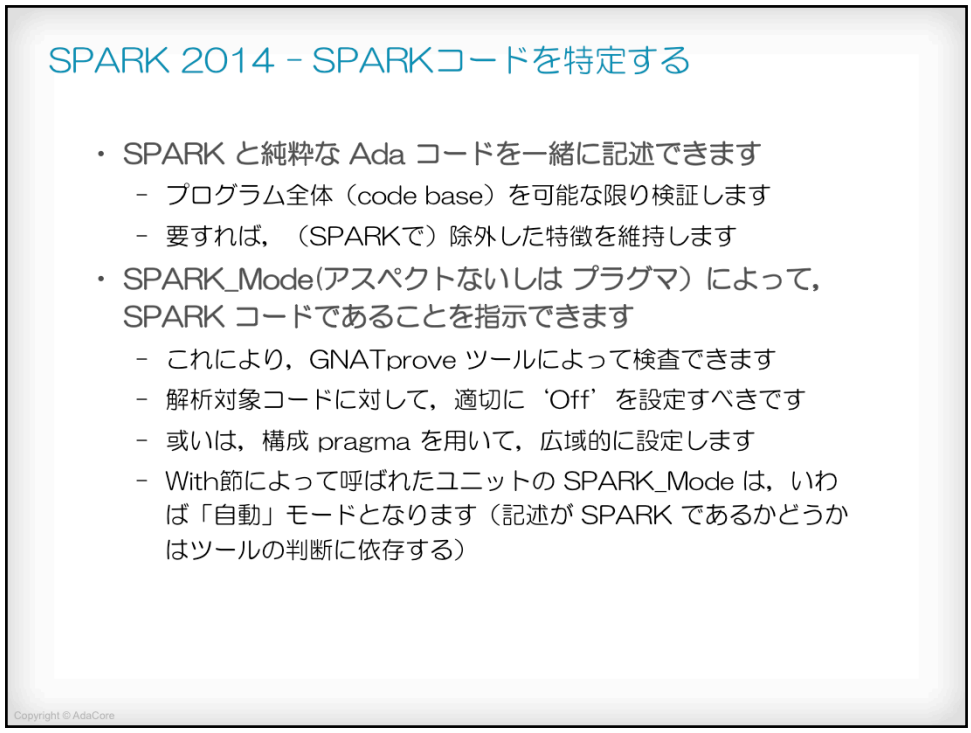

この Move_To_Total の例では，引数 Source の値を広域変数 Total に加えています．

次に，Source をゼロにリセットしています．これは明らかにプログラマは，Total と Source が別名化している可能性を気にしていません．

よくあることです．

このサブプログラムは，SPARKとして正当です．検証において，SPARK 2014 ツールは，プログラマのように，Total とSource の間に別名化はないと考えます．

この仮定が正しいことを確認するために，ツールは，Move_To_Total に対する全ての呼び出しにおいて，別名化が生じていないことを確認します．




SPARK言語は，仕様記述と検証が容易にできるように，プログラムの記述に制限を加えています．

しかし，ときには，これら制限内にいることができない，或いは制限を変えたいと望む場合があります．

SPARK 2014 ツールでは，SPARK コードであると識別できたコードにのみ適合性検査を行います．

この指定は，SPARK_Mode というアスペクトにより行います．

もし，明示的に記述されていなければ，SPARK_Mode は Off です．即ち，プログラムは，Ada コードと考えます．

このデフォルト設定は，構成プラグマ （configuration pragma）によって変更することができます．

既存の Ada ライブラリを使いやすくするために，SPARK_Mode の指定がなく，with節によって宣言されたエンティティでも，SPARK コードから利用できます．

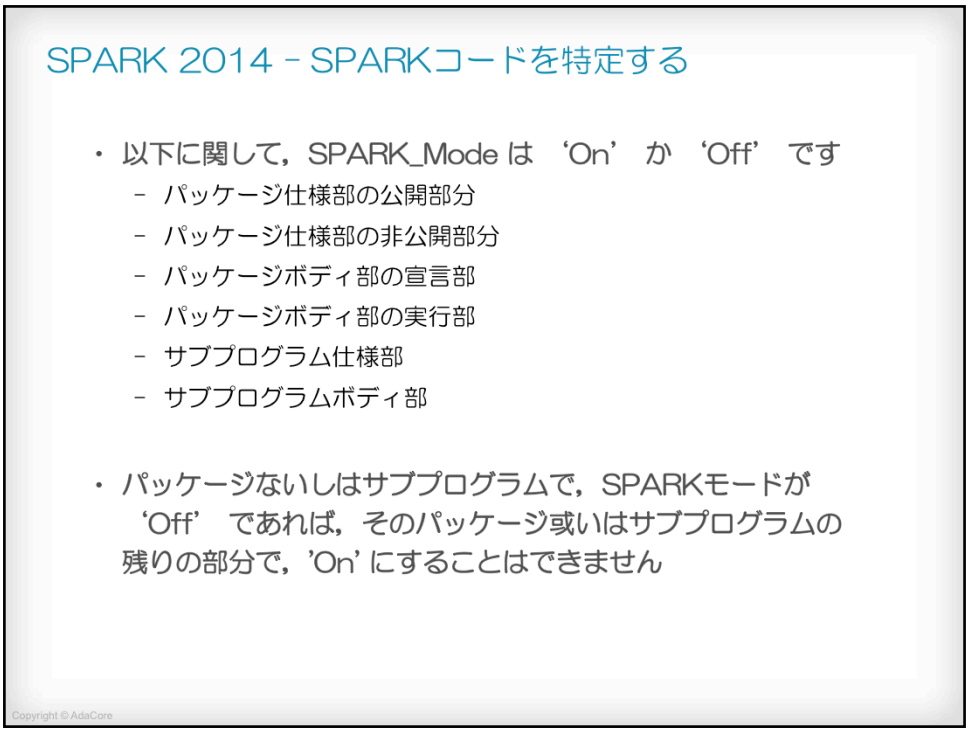
ツールは，SPARK コードを用いた箇所の宣言に対して，SPARK への適合性のみを効果的に検査します．

（訳注）ユニット（Unit） コンパイル単位．ただし，Ada 言語では，ボディ部を副単位として，別にコンパイルすることができる（他の言語には余りない特徴になります）．


これは，SPARK_Modeに関して，よくある使い方です．

パッケージ P は，仕様部が SPARK サブセットであるというエンティティのみを定義しています．

しかし，ボディ部は，（SPARK_MODE がオフであることから）完全な Ada 言語の記述であり，（SPARK として）解析されません．




SPARK_Mode は，ユニット単位で，細かく設定することができます．

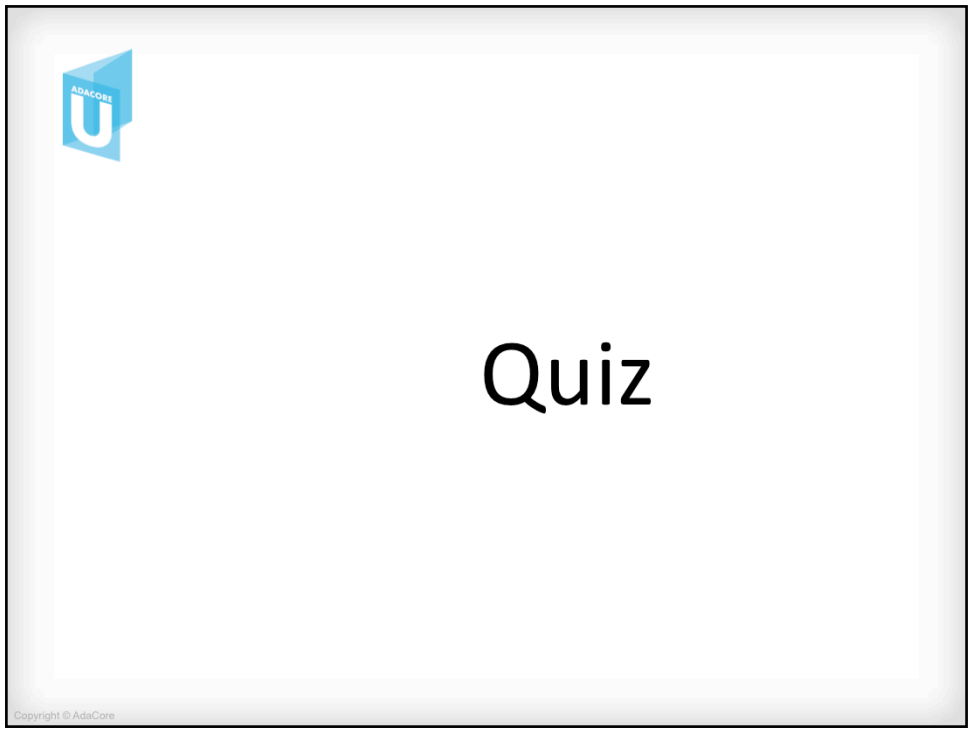
正確に言うと，つぎのようになります．パッケージは，４つの部分に分けることができます．仕様部の公開部分・非公開部分，本体部の宣言部（declarative part）・実行（statement part）部です．

４つのそれぞれで，SPARK_Mode をオン・オフすることができます．

同様にサブプログラムにも，仕様部とボディ部があります．

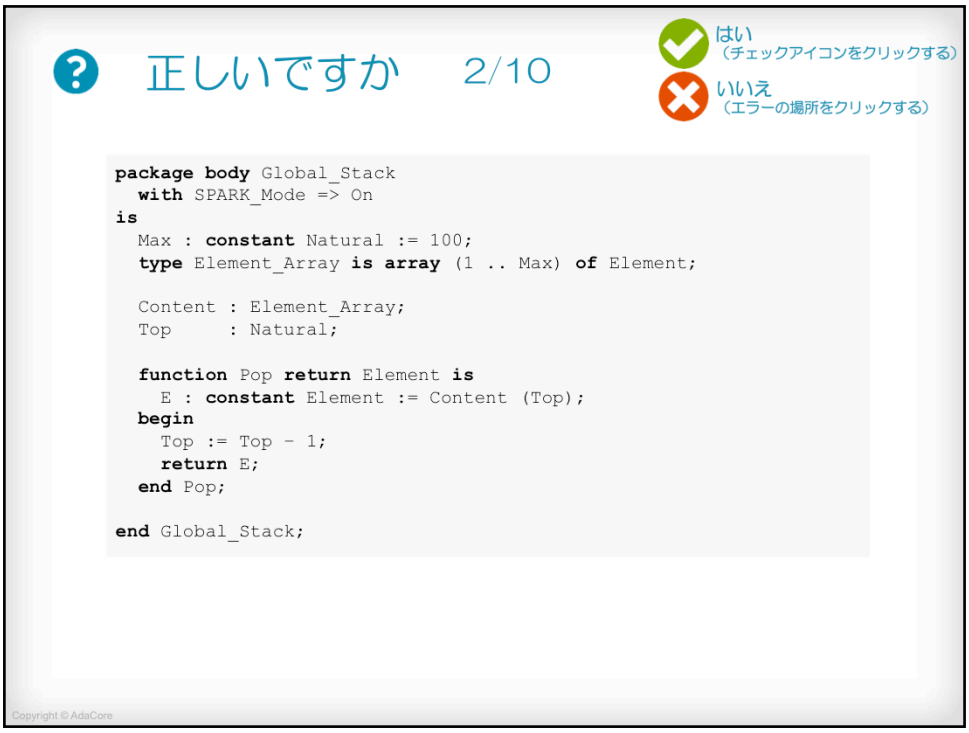
SPARK において，もっとも重要なのは SPARK_Mode が，オフにセットされたら，二度とオンにはできないということです．

SPARK_mode がオフのユニットに属するサブユニットにおいて，SPARK_mode をオンにすることはできません．同様に，ある箇所がオフであったときに，それに続く場所をオンにすることもできません．


Quiz
Copyright © AdaCore

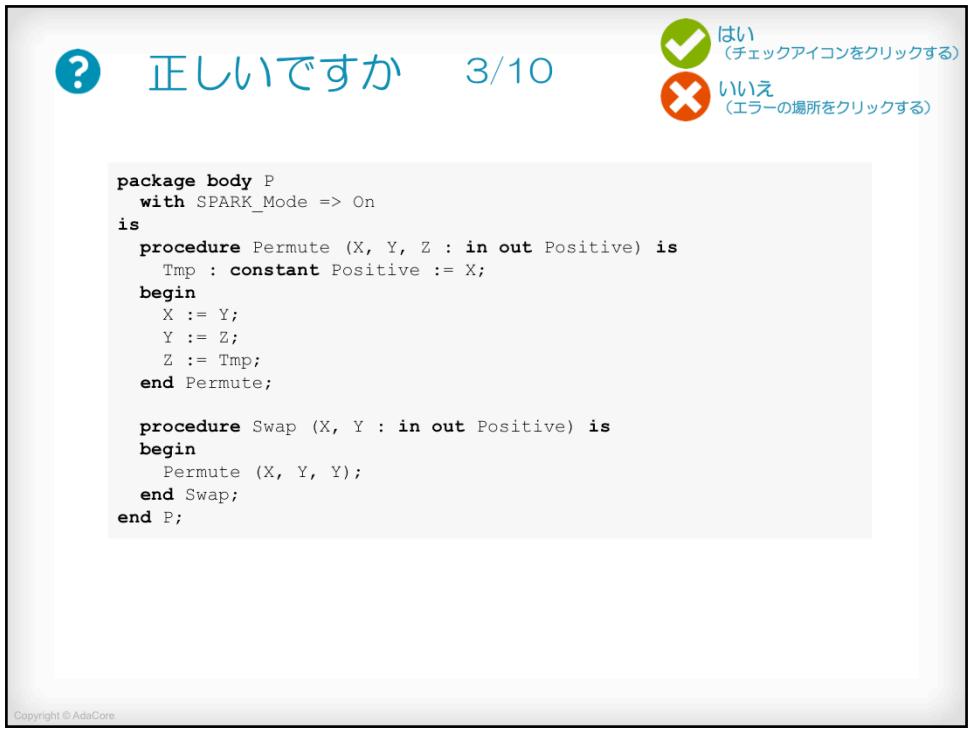

これは，Stack 型を定義しているパッケージです．Element 型の要素を持ち，関数は通常のスタックに求められる機能を持ちます．ここでは，SPARK サブセットと指定しています．


SPARK では，副作用のある式を用いることはできません．Pop 関数は，パラメータ S を更新できません．

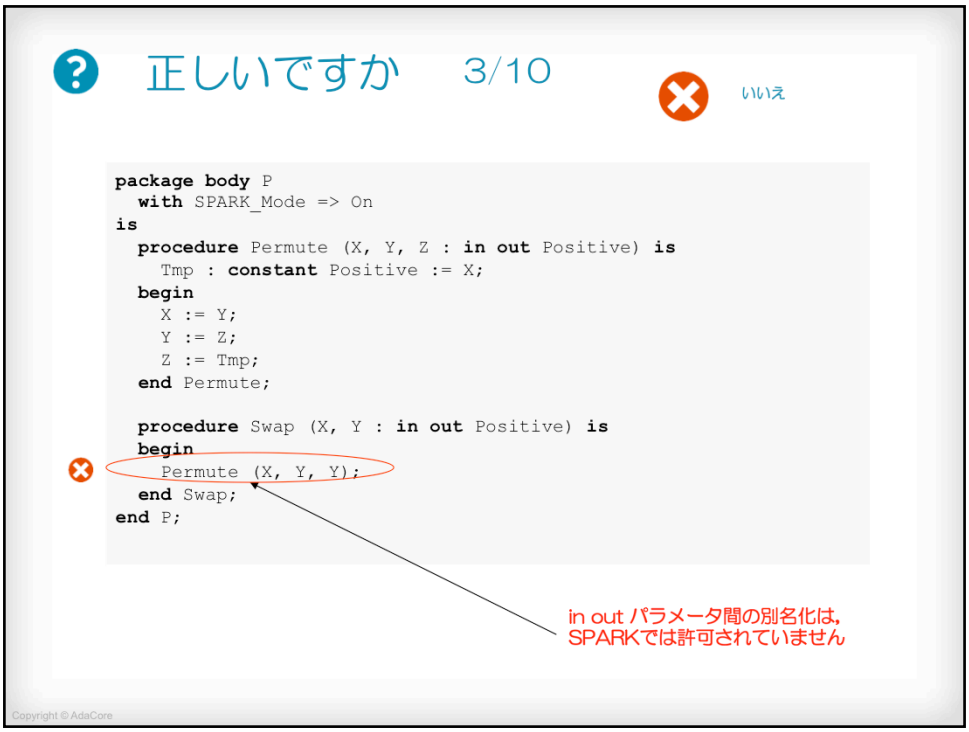



ここで，単一のスタックインスタンスを持つパッケージのボディ部に着目します．Content と Top はスタック状態を記録するための広域変数です．もういちど，注意してください．このパッケージは SPARK サブセットです．

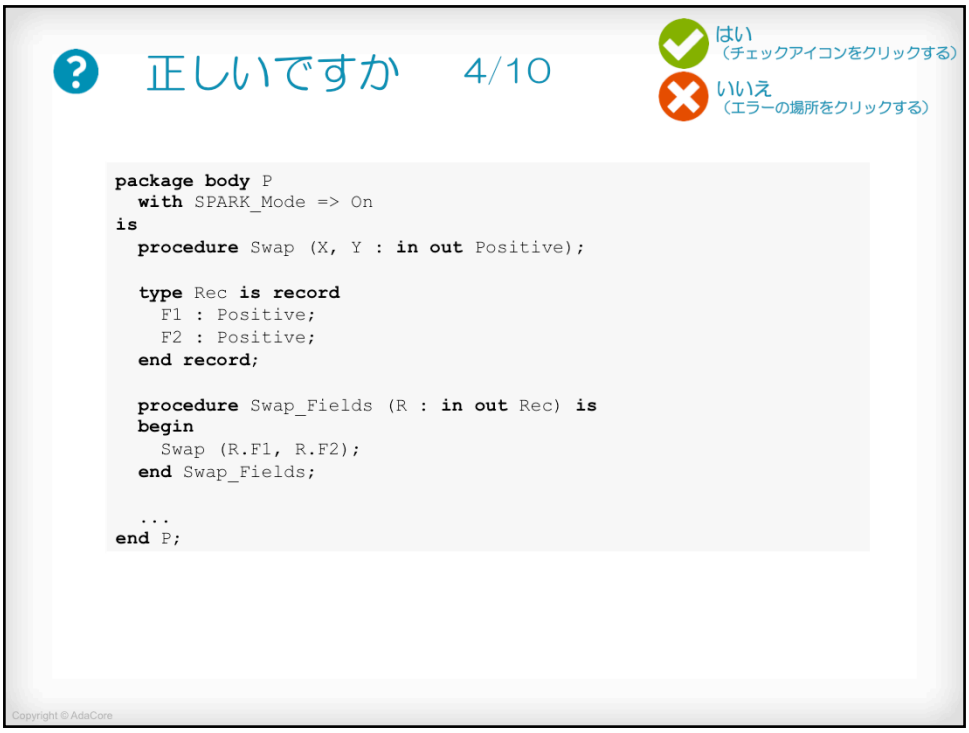

先の問題と同様に，関数は，副作用と無縁であるべきです．ここでは，Pop 関数内で，広域変数 Top を更新しています．これは，SPARK では許されていません．



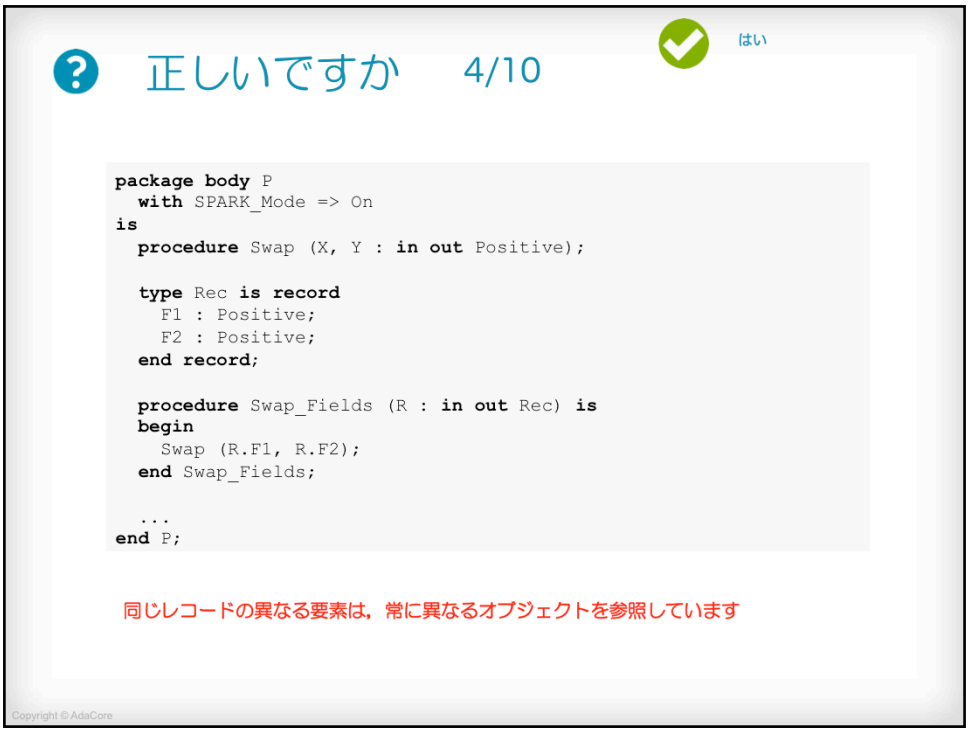

２つの手続き，Permute と Swap があります．Permute は，3つの引数の値に環状の入れ替えを行います（巡回置換：即ち右隣の値を左へ，Xの値はZに）．Swap は，X と Y の値を交換（swap）するのに，手続き Permute を用います．



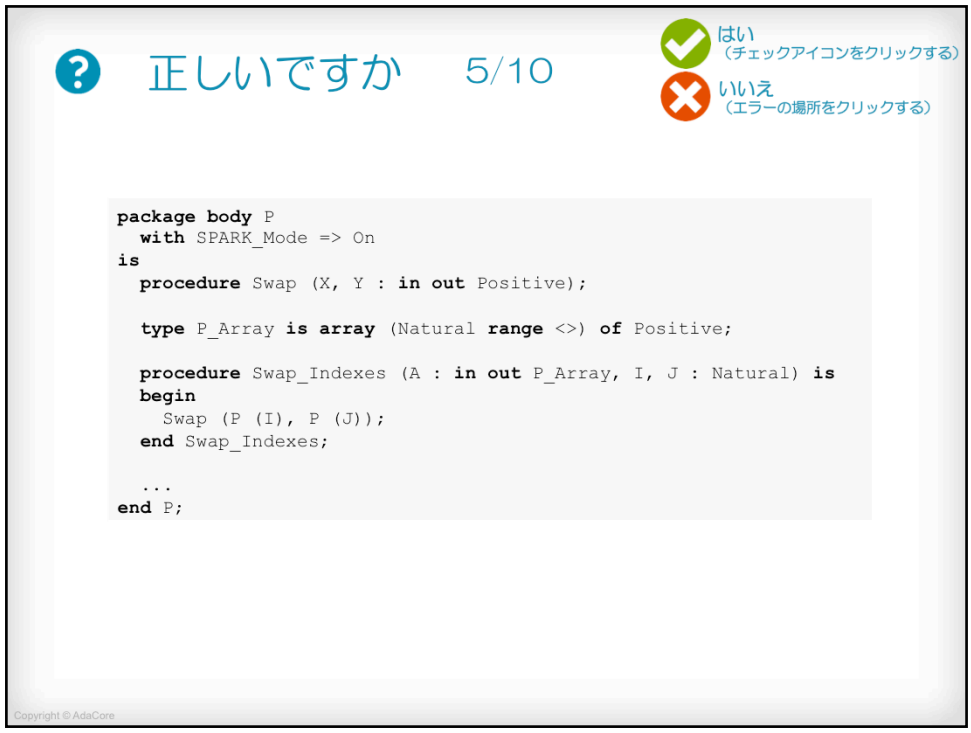

Permute の呼び出しにおいて，Y と Z の値は別名化しています．これは，SPARK では許されていません．この例は，どうして SPARK で別名化が許されていないのかを示しています．Y と Z は正の整数なので，コピーによって受け渡されます．Permute 呼び出し後の結果は，呼び出し後のコピー順に依存します．



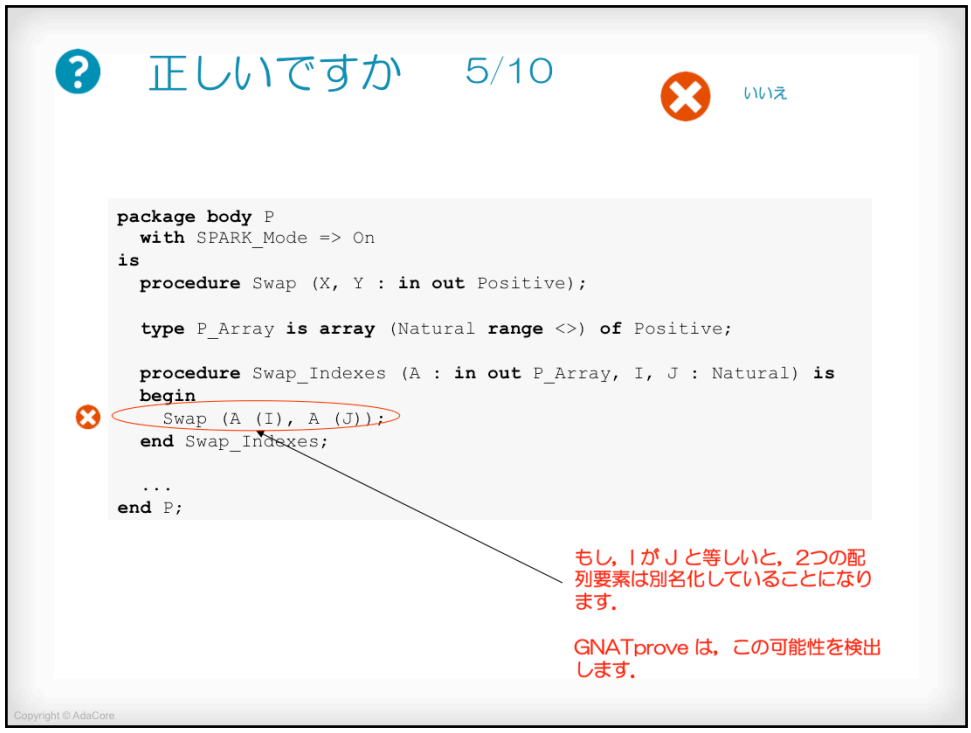

ここで，R の２つのレコード型の要素を交換するために，Swap 手続きを用いています．



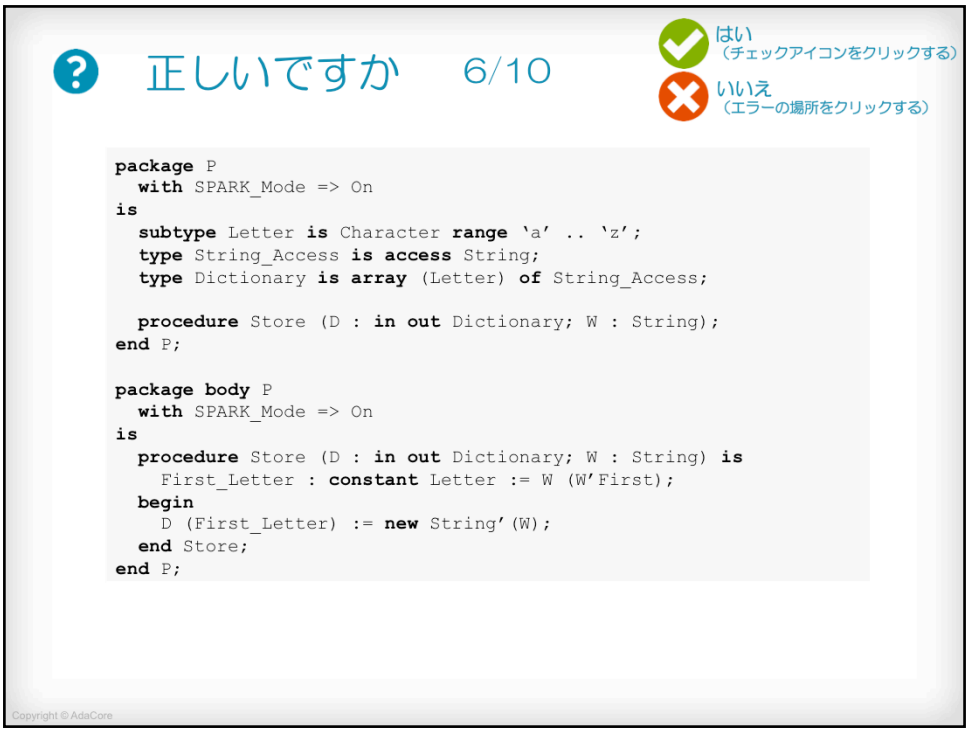

これは正しいコードです．Swap の呼び出しは安全です．レコードオブジェクトの二つの異なる要素は，同一のオブジェクトを参照していません．



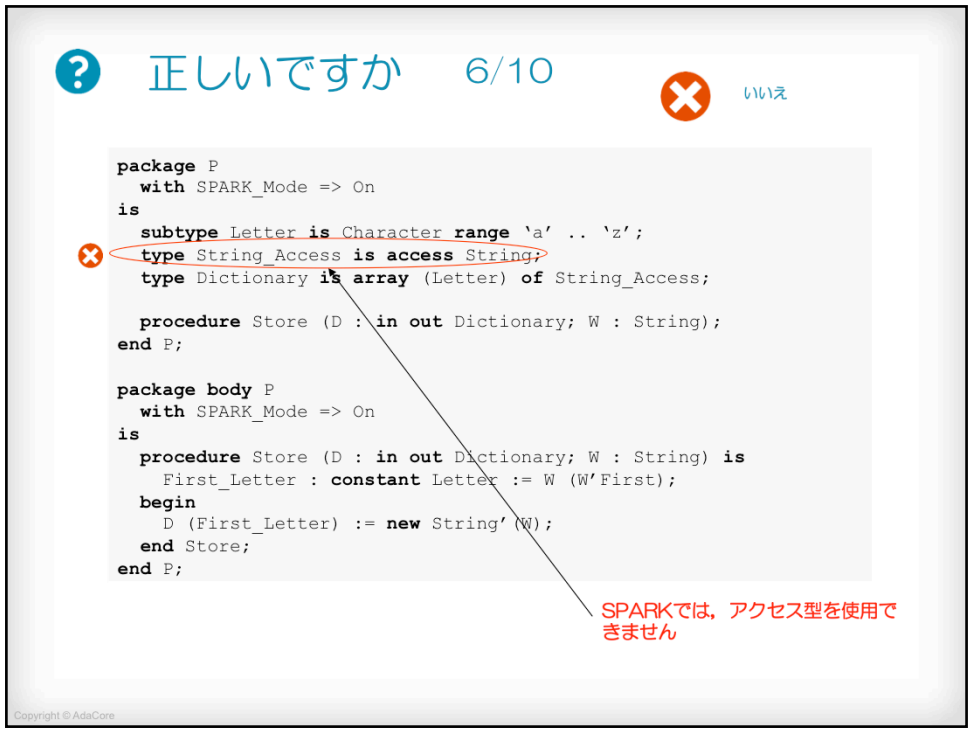

先の例題を，レコード型ではなく，配列を用いて少し変更しています．Swap_Indexes は，配列 A の値を交換するために，Swapを用いています．



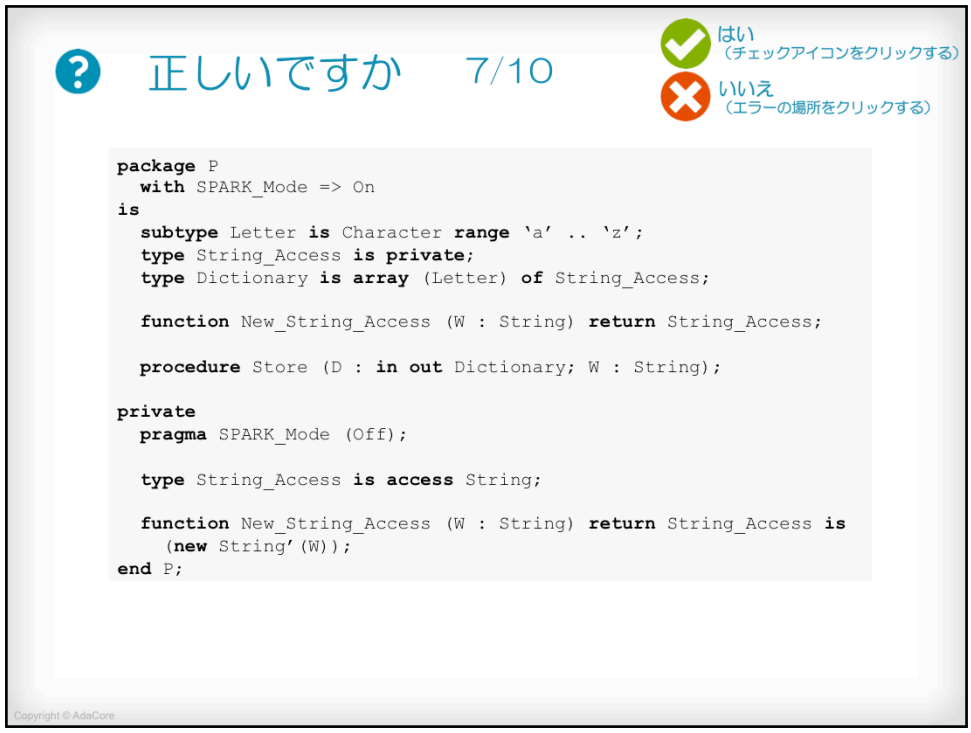

レコードの例とは違い，Swap を呼び出すときに，要素 A (I) と A (J) が，同一ではない，ということを知ることができません．



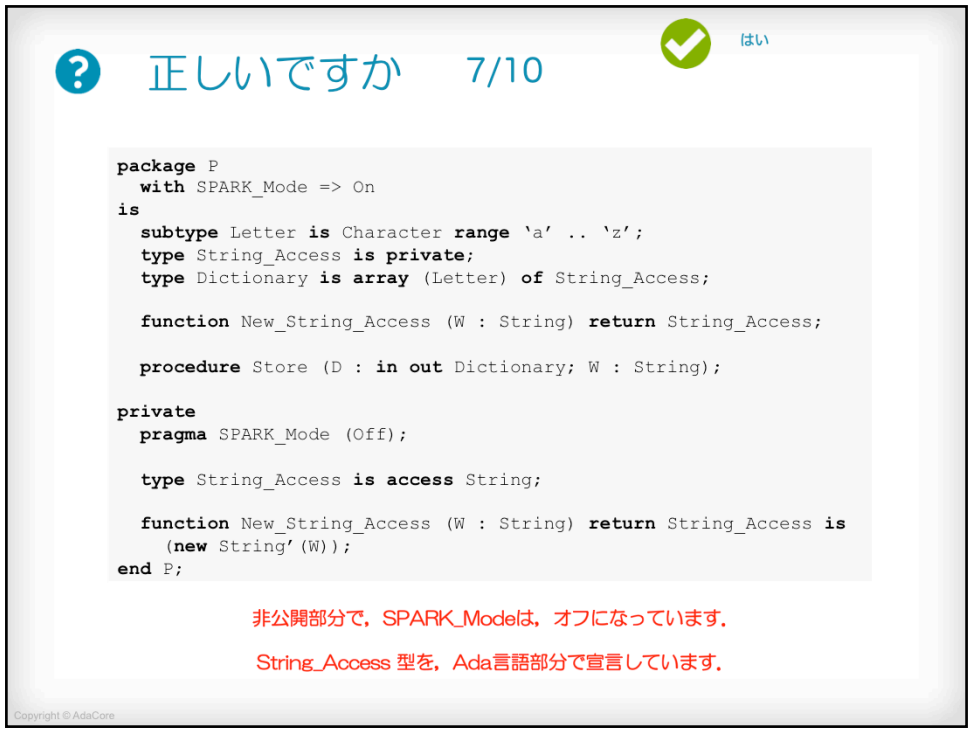

これは，Dictionary 型を宣言しているパッケージです．文字毎に単語を格納している配列です．手続き Store は，辞書中の正確なインデクスに単語を挿入することができます



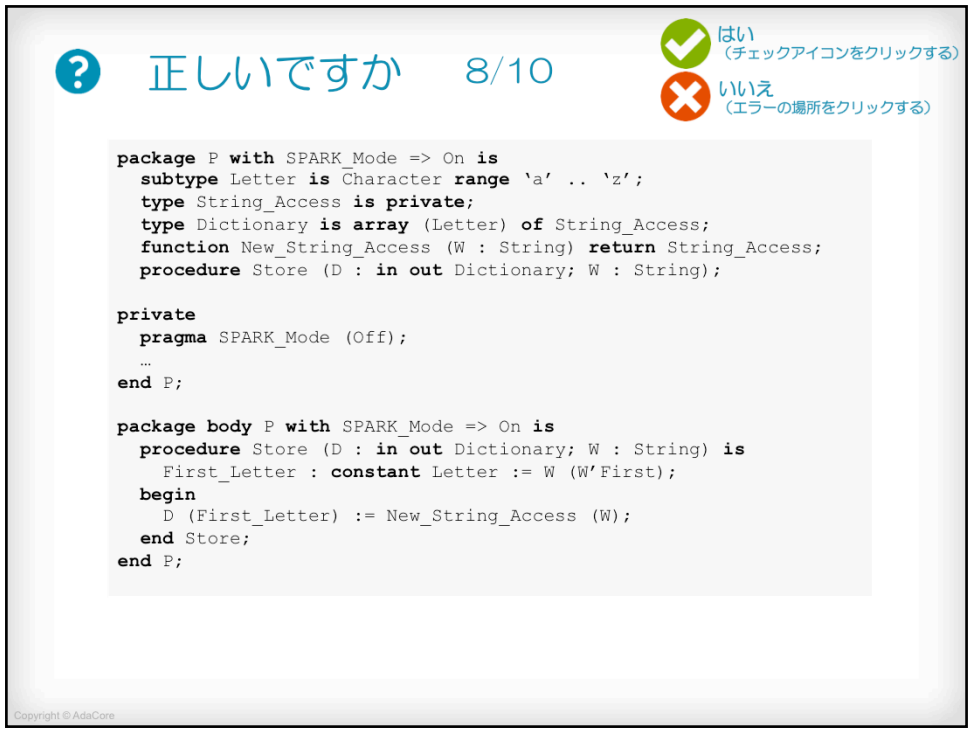

このコードは正しくありません．アクセス型は，SPARK には含まれていないからです．しかし，この例の場合，書き方は有効で，アクセス型なしに任意長の文字列を配列に格納することはできません．このため，SPARK_Mode で，アクセス型を使用している部分を分離します．これは，SPARK_Mode を用いた細かな設定になります．



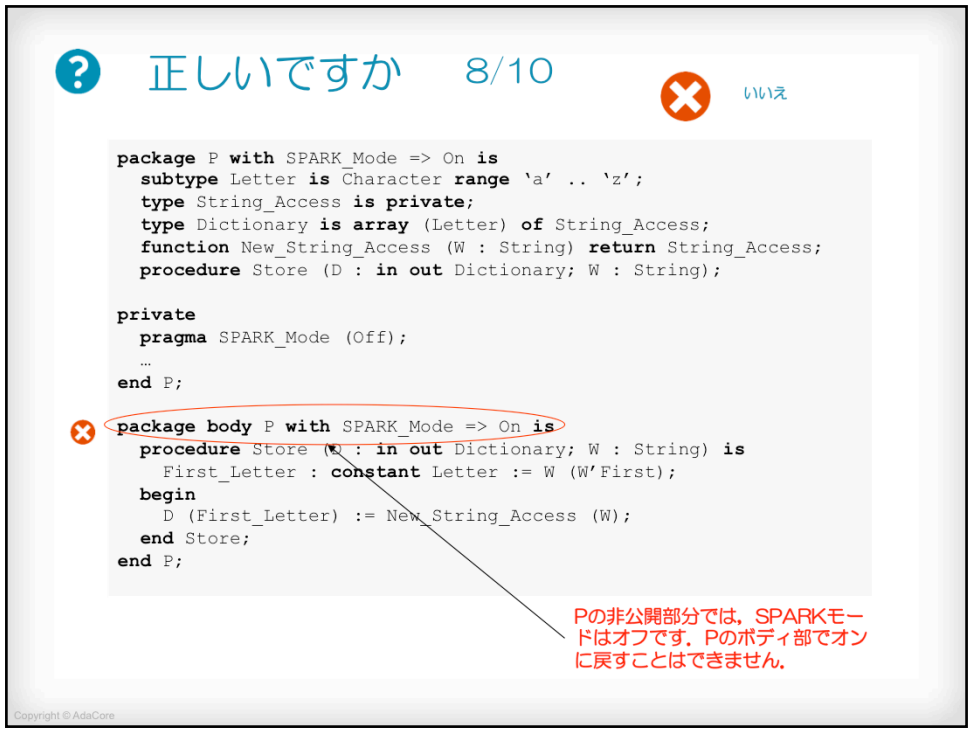

これは前ページの例を修正したプログラムになります．ここでは，アクセス型を P の非公開部分に隠しています．



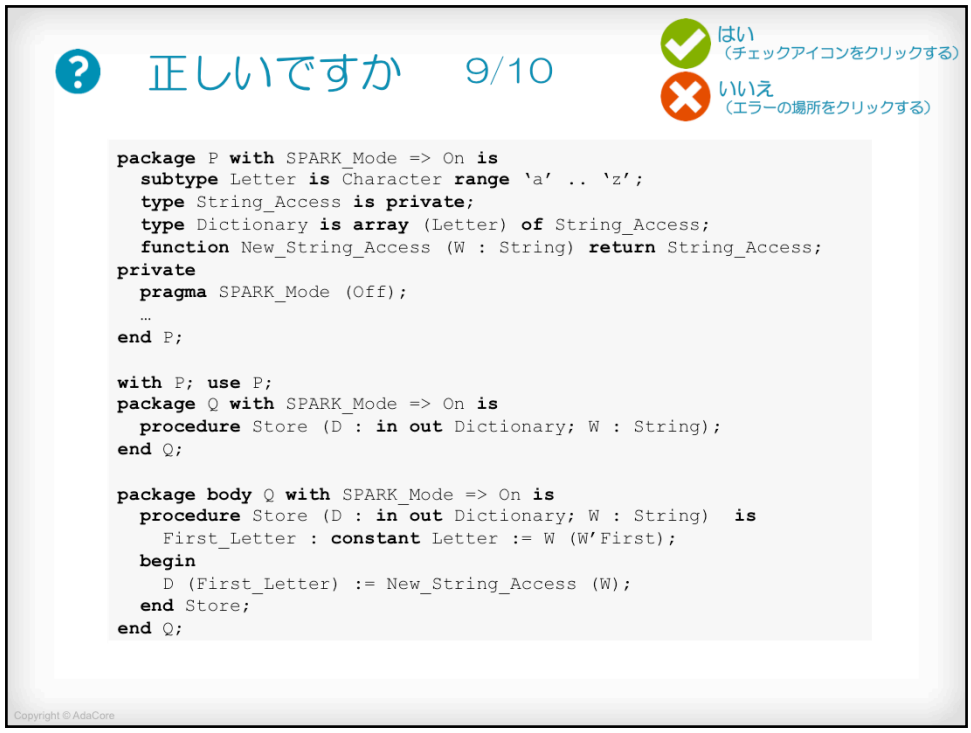

Ada 言語部分で，アクセス型を定義し利用しているため，これは正しいコードとなります．



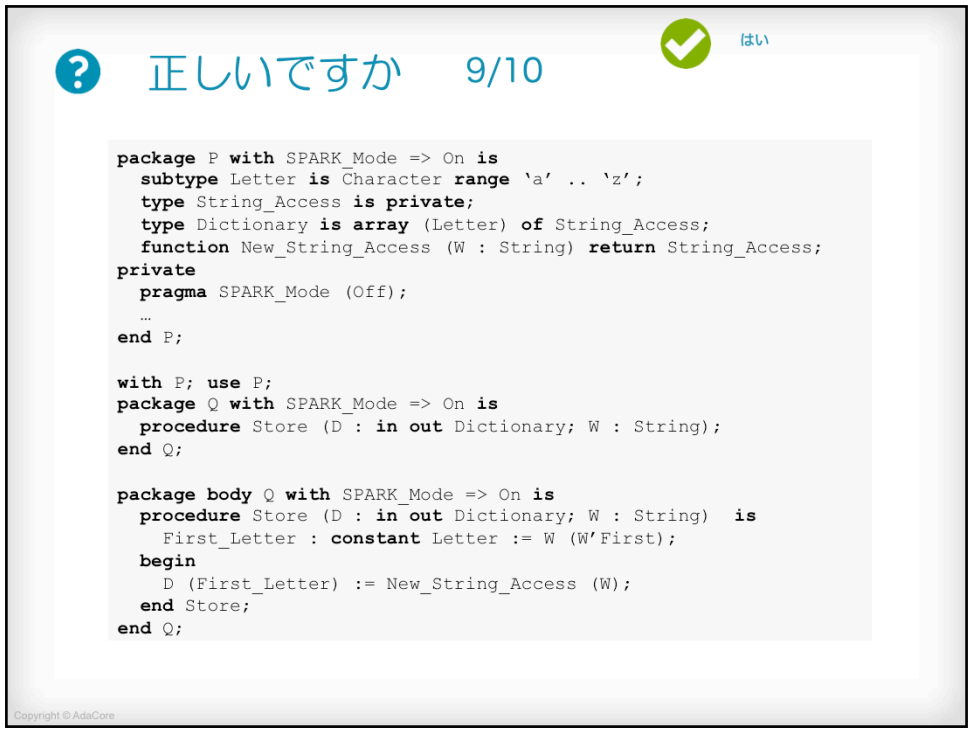

手続き P のボディ部を，次に見ます．ここには再度 Store の定義があります．



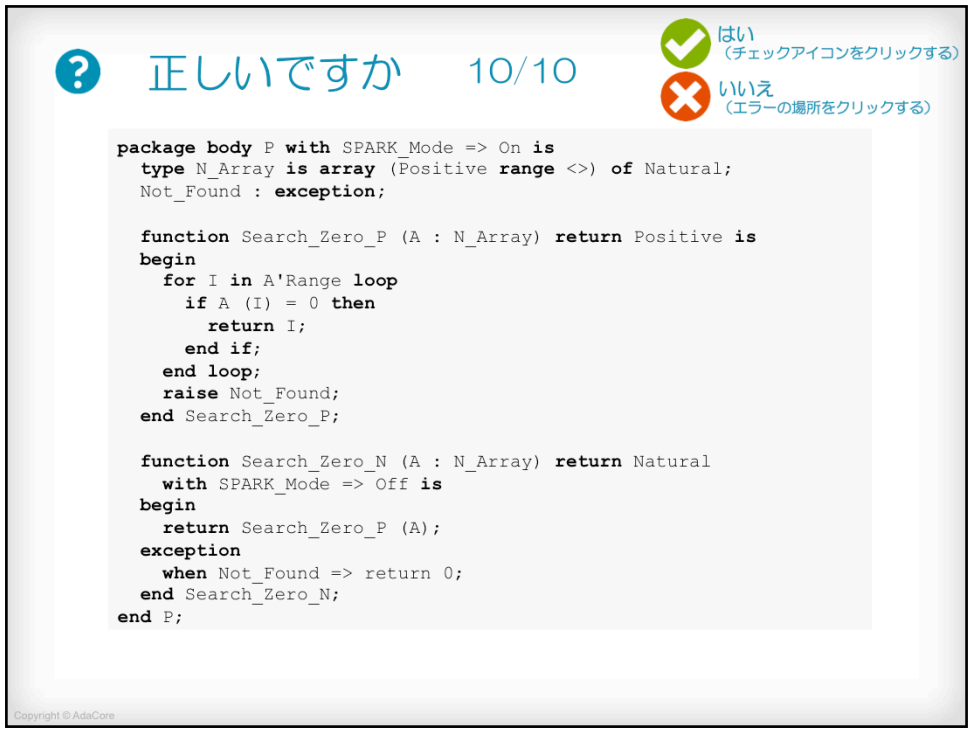

Store のボディ部では，SPARK 言語外の要素を用いていませんが，SPARK_Mode を P のボディ部で用いることはできません．実際に，用いていないときでさえ，完全な Ada である P の非公開部分に対して，可視性があるからです．



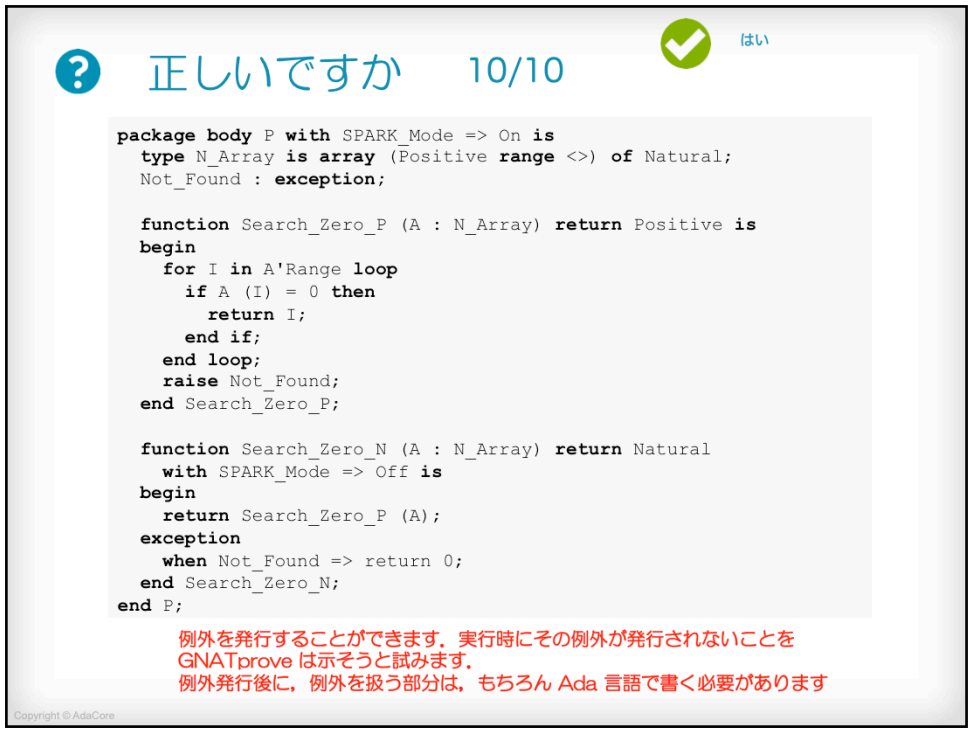

ここで，Store の宣言と手続きボディ部を，Q という名前の別のパッケージに移動しました．




このプログラムは，まったく問題がありません．SPARK言語で多くの部分を書きつつ，アクセス型を利用できています．従って，GNATprove は，このプログラムを解析することが可能です．




ここには，２つの関数があります．配列 A 中のゼロを探索する関数です．最初の関数は，ゼロが見つからない場合に，例外を発行します．もう一つの関数は，単にゼロを返します．




これは完全に正しいコードです．GNATprove　は，Search_Zero_P 中で，Not_Found 例外が決して発行されないことを示すように試みます．Search_Zero_N をみてください．そのようなプロパティが真ではない可能性があります．従って，ユーザは，Not_Foundが，適切であるときにのみ発行されることを検証する必要があります．


ADACORE
university.adacore.com
Copyright © AdaCore