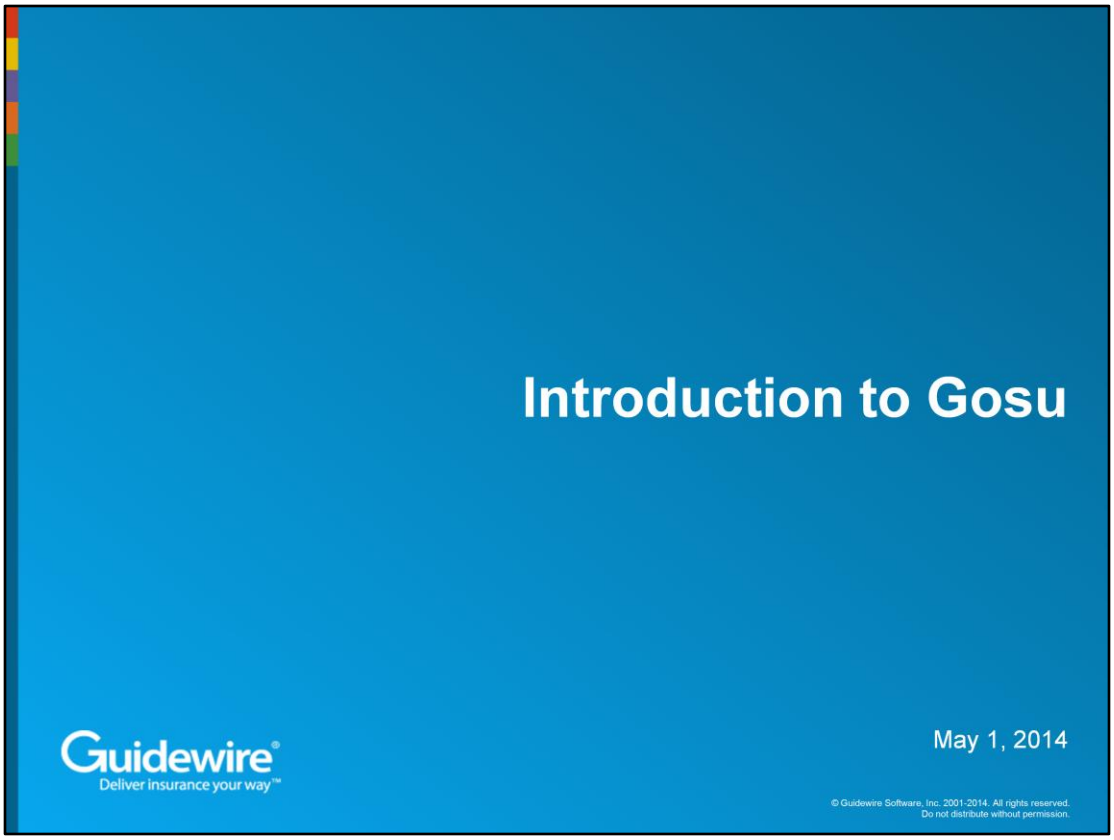


Guidewire training materials contain Guidewire proprietary information that is subject to confidentiality and non-disclosure agreements. You agree to use the information in this manual solely for the purpose of training to implement Guidewire software solutions. You also agree not to disclose the information in this manual to third parties or copy this manual without prior written consent from Guidewire. Guidewire training may be given only by Guidewire employees or certified Guidewire partners under the appropriate agreement with Guidewire.

Lesson objectives

- By the end of this lesson, you should be able to:
  - Describe the ways that Guidewire applications use Gosu
  - Write Gosu using basic Gosu syntax
  - Describe Studio features that aid in the writing of Gosu

This lesson uses the notes section for additional explanation and information.
To view the notes in PowerPoint, select View → Normal or View → Notes Page.
When printing notes, select Note Pages and Print hidden slides.

Lesson outline

- Gosu overview
- Gosu Scratchpad
- Gosu statements
- Gosu objects
- Gosu subtypes

Guidewire Gosu

- Guidewire's programming language
  - Similar to Java and compatible with Java
  - Has elements of both procedural and object-oriented programming languages
- Executes fundamental application behavior
- Manages complex business processes
- Executes hierarchical business rules
- Specifies dynamic user interface behavior

Gosu is Guidewire's open-source, publicly available programming language. Gosu has elements of both procedural and object-oriented programming languages and is similar to JavaScript and Java.

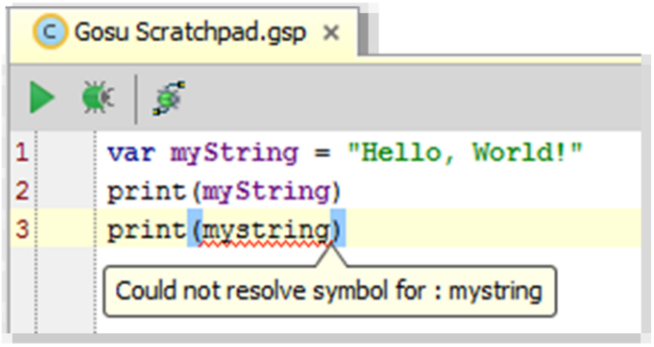
Guidewire developed Gosu for several reasons.

First, there was a desire to have a single syntax that could be used to work with all of the elements relevant to Guidewire products (such as entities, display keys, classes, class enhancements, Java classes, permissions, and script parameters) even though these items have fundamentally distinct internal implementations. There was no existing language that provided this ability.

Second, there was a desire to have code auto-complete features in Studio. This is possible only with a statically typed language. Most scripting languages, such as JavaScript, Perl, Python, and Ruby, are dynamically typed.

Prior to ClaimCenter 6.0, PolicyCenter 4.0, and BillingCenter 3.0, Gosu was known as GScript.

Gosu is now case-sensitive



- Previous versions of Gosu were case-insensitive
- Small, but noticeable performance compiler cost for case-insensitivity

- Refer to variables, types, and symbols using exact casing
- Compiler warns about not being able to resolve a symbol

Gosu in Guidewire applications (1)

Business rules

- Apply specific programming logic for testing a condition and performing an action
- Examples:
  - Event Fired Rules
  - Pre-update Rules
  - Validation Rules

Entity enhancements

- Extend entity functionality with programming logic as entity methods
- Examples:
  - ABCompany.maskTaxId() returns a value with a mask for the first 5 characters

A rule is a single decision in the following form of testing a condition and then taking an action: if {some conditions} then {take some action}. A rule set combines many individual rules into a useful set to consider as a group.

Gosu enhancements provide additional methods (functionality) on a Guidewire entity. For example, you can create an enhancement to the ABContact entity. Within enhancement, you add methods that support new functionality. Guidewire Studio will show the new method for an entity of the type ABContact.

Gosu in Guidewire applications (2)

Entity names

- Programming logic that defines how to display a name for an entity instance
- Examples:
  - Drop-down lists of contacts
  - DisplayName property of entity instance

Gosu classes

- Encapsulate data and code for a specific purpose or function
- Examples:
  - Utility helper classes for logging or string functions
  - Plugins for integrations with other applications

The Entity Names lesson covers how to create, modify, use, and deploy entity names. The Gosu Classes lessons covers how to create, modify, reference, and deploy Gosu classes.

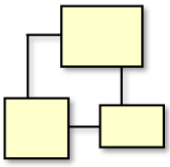
Gosu in Guidewire applications (3)

PCF methods

- Specify user interface layout and dynamic behavior
- Examples:
  - Set the visible property of the a DateInput widget when the value changes in a BooleanDropdown widget

Workflows

- Run custom business processes asynchronously, optionally with multiple states that transition over time
- Examples:
  - Renewal and cancelation workflows

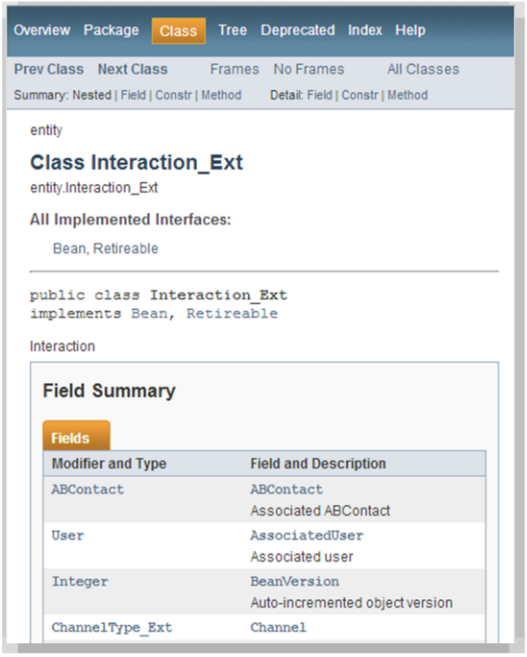


A workflow executes processes whose execution is time-based or is triggered by events in external applications, or is both triggered and time-based.

Workflows are available in all three primary Guidewire applications. PolicyCenter uses workflows to manage policy transactions (such as renewals). BillingCenter uses workflows to manage account delinquency and agency bill cycles. Although the feature is available in ClaimCenter, the base configuration of ClaimCenter does not use workflows.

Gosu API Reference

- Contains navigable documentation for
  - Entities
  - Typelists
  - Display keys
  - Classes
  - Packages
- **`gwXX regen-gosudoc`**
  - Run from bin command window where XX is application code
  - Output is to **`…\<ApplicationRoot>\build\gosudoc\index.html`**

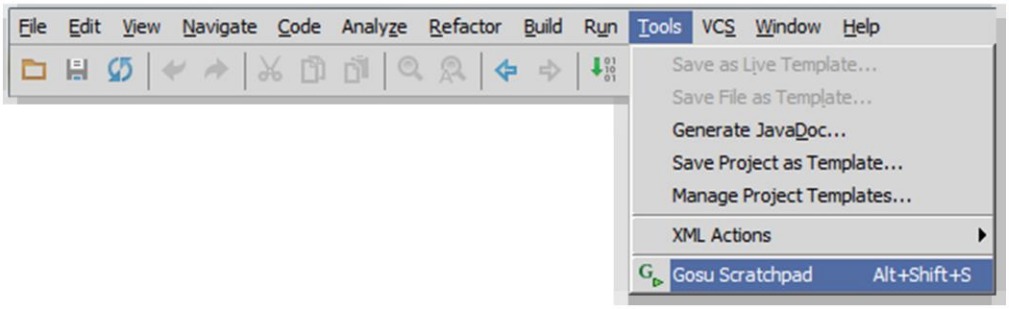


You can regenerate the Gosu API Reference from the command window of the bin directory. Use the command gwXX regen-gosudoc where XX is the application two letter abbreviation.

Lesson outline

- Gosu overview
- Gosu Scratchpad
- Gosu statements
- Gosu objects
- Gosu subtypes

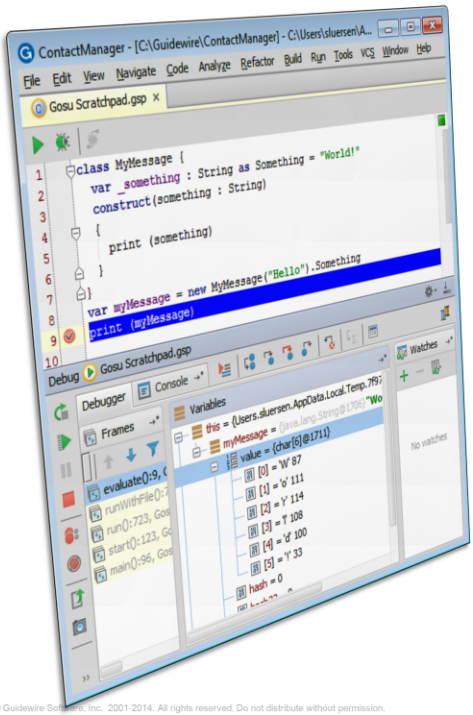
Start Gosu Scratchpad



- Main menu → Tools → Gosu Scratchpad
- ALT+SHIFT+S
- Opens code editor to write, run, and debug Gosu code

Gosu Scratchpad is a Guidewire Studio code editor that lets you write, run and debug Gosu code.

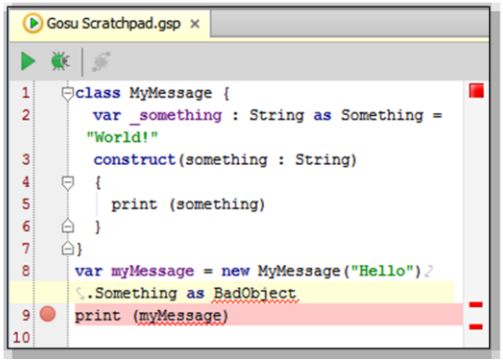
Gosu Scratchpad



- Write, execute and debug Gosu code
- Common editor features supported
  - Gutter Area
  - Smart completion
  - Validation bar
- Consists of
  - Code editor
  - Debug window
  - Run window

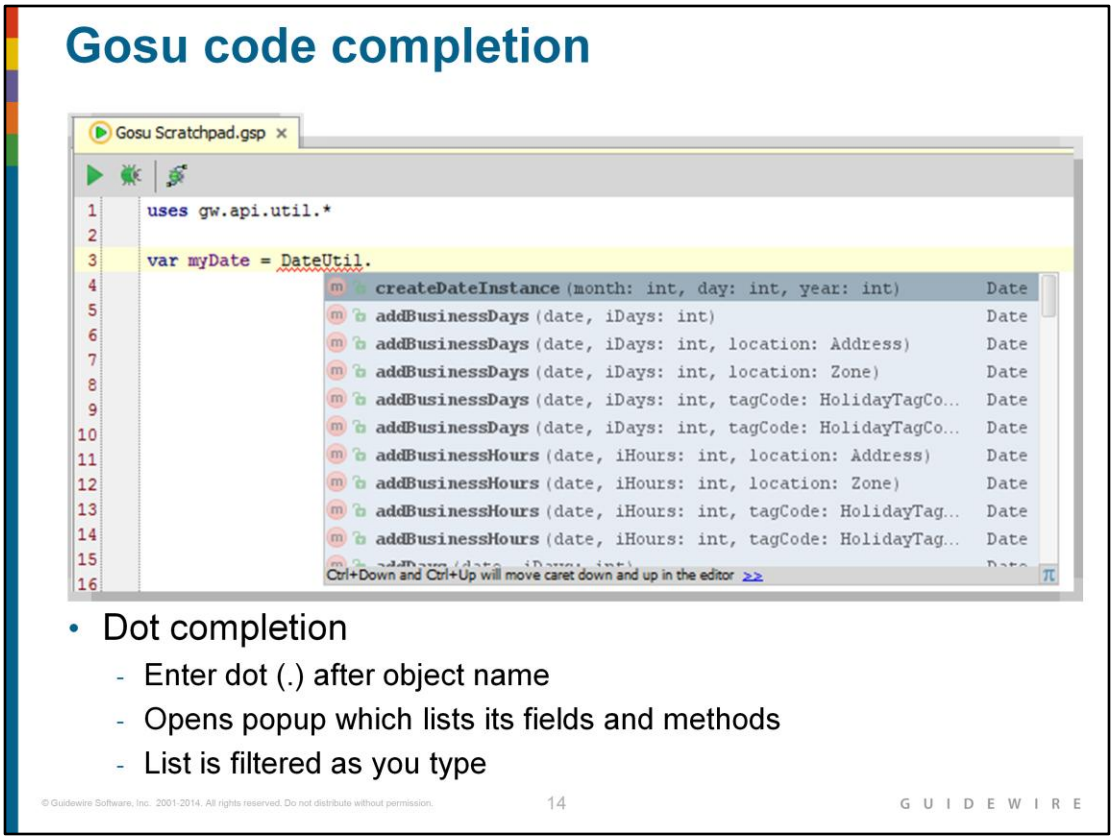
You can use Gosu scratchpad to run and debug Gosu code in Gosu Scratchpad or in to debug code in the Run Debug Process for the given Guidewire application project.

Gosu Scratchpad: code editor



- Editor toolbar
  - Run and Debug
  - Debug in application process
- Gutter
  - Show Line Numbers, Indent Guides, and Use Soft wraps
  - Set Break points
- Validation bar
  - Red marks syntax error
- Code completion
  - Basic and smart
- Context Menu
  - Paste Java as Gosu

Guidewire Studio provides several different ways of obtaining information about application objects and APIs to assist you in writing valid rules and Gosu code.



You can use Guidewire Studio to complete your code by placing your cursor at the end of a valid object and typing a dot (.).

The dot notation action causes Studio to open a pop-up window that contains all the properties, methods, and objects that are possible for this context. As you type, Studio filters this list to include only those choices that match what you have typed thus far. Use the down arrow to highlight the item you want and press Enter or Tab to select it. If the selection is not the initial default selection, you can also use the Space bar.

Gosu code help

Keystrokes

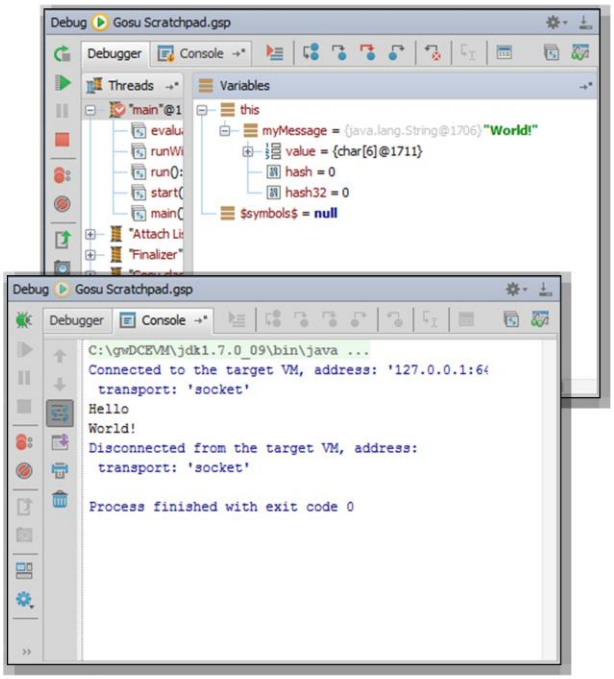
- Code
  - ALT+ ENTER
    - Import class / create method
  - CTRL+SPACE
    - Import class / suggest
- Information
  - CTRL+P
    - Show arguments for method
- Navigate
  - CTRL+B
    - Find symbol declaration

SmartHelp

- Light bulb
  - Icon indicates that there is help with code completion

Debug tool window

- ALT+5
  - Opens Debug window
- Debugger tab
  - Rerun, Resume, Pause, Stop
  - View breakpoints
  - Step over, into and out
  - Inspect and watch
- Console tab
  - View output



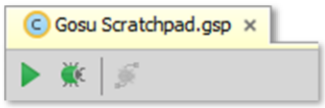
The debugger enables you to execute your application step by step, examine program information related to variables, watches, or threads, and change your program without leaving Guidewire Studio.

Run tool Window

- ALT+4
  - Opens Run window
- View Run 'Server' application log
  - Same as console window output
  - View console output
- Gutter commands
  - Stop server, (Re)run server, Pause Output, Exit run

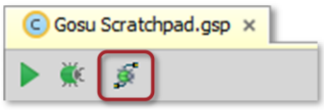
API references in Scratchpad

No Guidewire API data backed references

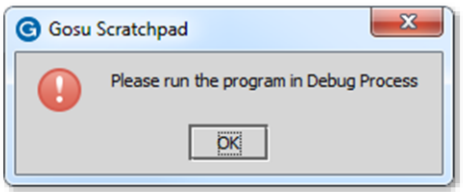


- Run
  - Execute Gosu code without starting application
  - Output to Run window
- Debug
  - Set and hit break points
  - Execute Gosu code without starting application
  - Output to Debug window

Guidewire API data backed references



- Run in Debug Process
  - Requires Debug 'Server' running
  - Output to Debug window
  - Cannot hit break points in Scratchpad



18
GUIDEWIRE

To execute code in Gosu Scratchpad.gsp, open Gosu Scratchpad using Main menu → Tools → Gosu Scratchpad or with the keystroke ALT+SHIFT+S.

If the code in Gosu Scratchpad does not reference the data backed Guidewire API, it is not necessary to execute the code within a debug process (Run in Debug 'Server' process). In the Gosu Scratchpad toolbar, click Run to execute the code and view the console output in the Run tools window. You can also select Run 'Gosu Scratchpad.gsp' from the main menu of Studio: Main menu → Run → Run 'Gosu Scratchpad.gsp'. To debug, click Debug and view the console output on the Debug tools window. You can also select Debug 'Gosu Scratchpad.gsp' from the main menu of Studio: Main menu → Run → Debug 'Gosu Scratchpad.gsp'.

If the Gosu code does reference the data backed Guidewire API, you must first start the server in debug mode so that you can execute the code in a debug process. To debug the server, select Main menu → Run → Debug 'Server' or Main menu → Run → Debug… → Server. In the Debug tools window, confirm that the application is running and is ready (***** AppName ready *****). Then, open Gosu Scratchpad using Main menu → Tools → Gosu Scratchpad or with the keystroke ALT+SHIFT+S. In Gosu Scratchpad toolbar, click Run in Debug Process.

Gosu Scratchpad compilers

Non-Guidewire application process

- Uses Gosu Community Release compiler

- Strict adherence to strong typing
  - int cannot be null
  - boolean cannot be null
- Available in Gosu Scratchpad when **NOT** running in debug process

Guidewire application process

- Uses Gosu Guidewire Platform compiler

- Coerces certain types behind the scenes
  - int coerced to 0 if null
  - boolean coerced to false if null
- Available when application and when Scratchpad is running in debug process

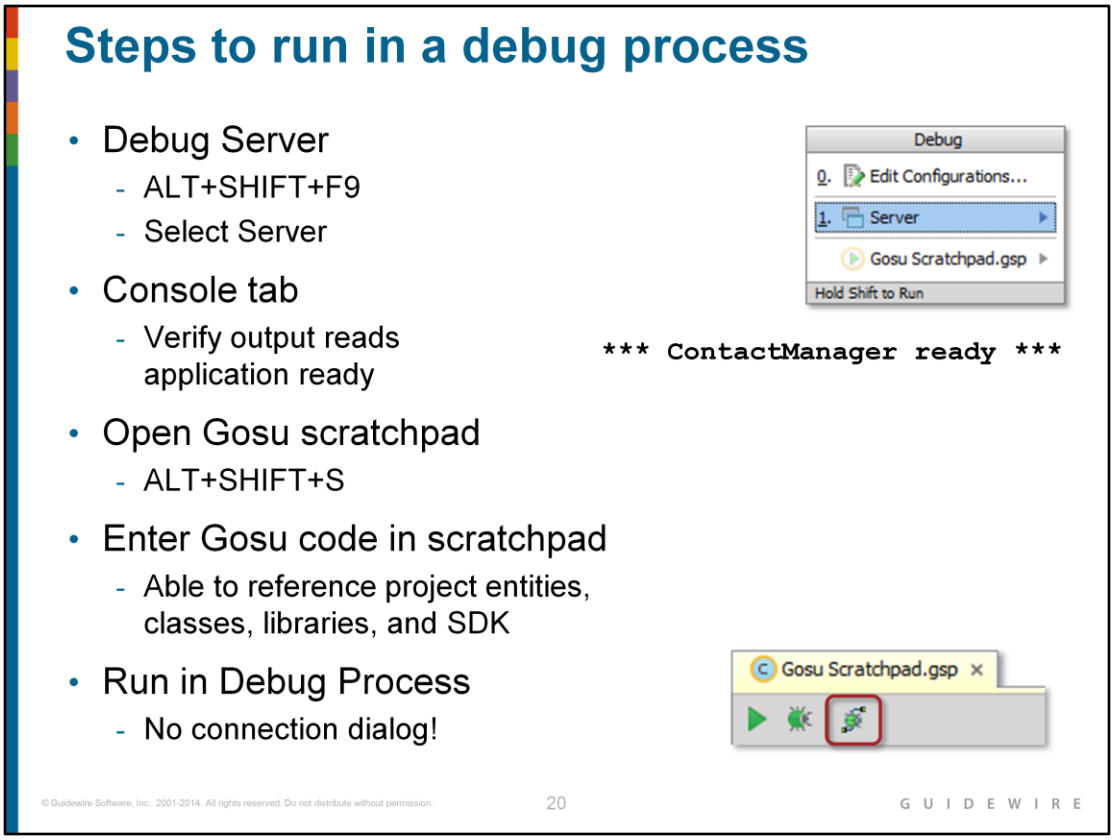
The Gosu Community Release is also known as the open source Gosu. Learn more about this version of Gosu at http://gosu-lang.org. The Gosu in the Guidewire Platform releases are incorporated into all Guidewire applications.

The coercion rules are different between the Platform and Community Release Gosu compilers. Generally speaking the community release rules are pretty strict and there are very few implicit coercions. Platform Gosu is more liberal and allows things such as a String being coerced to a decimal, for example.

Another difference between platform and community Gosu is with null safety of comparator operators. In Platform Gosu, the comparator operators such as "<" and ">" are null-safe. They will never generate a null pointer exception (NPE). In Community Release Gosu, the comparator operators such as "<" and ">" are NOT null-safe.

Also, the standard period "." operator for property access and method access works differently in an important way between Platform and Community Releases of Gosu. In Platform Gosu, the period operator is null-safe for properties. In Community Release Gosu, the period operator is NOT null-safe for properties.

In both Gosu versions, the new ?. operator is the null safe period operator for properties. Thus, in Platform Gosu it is redundant to compare to the regular period operator.



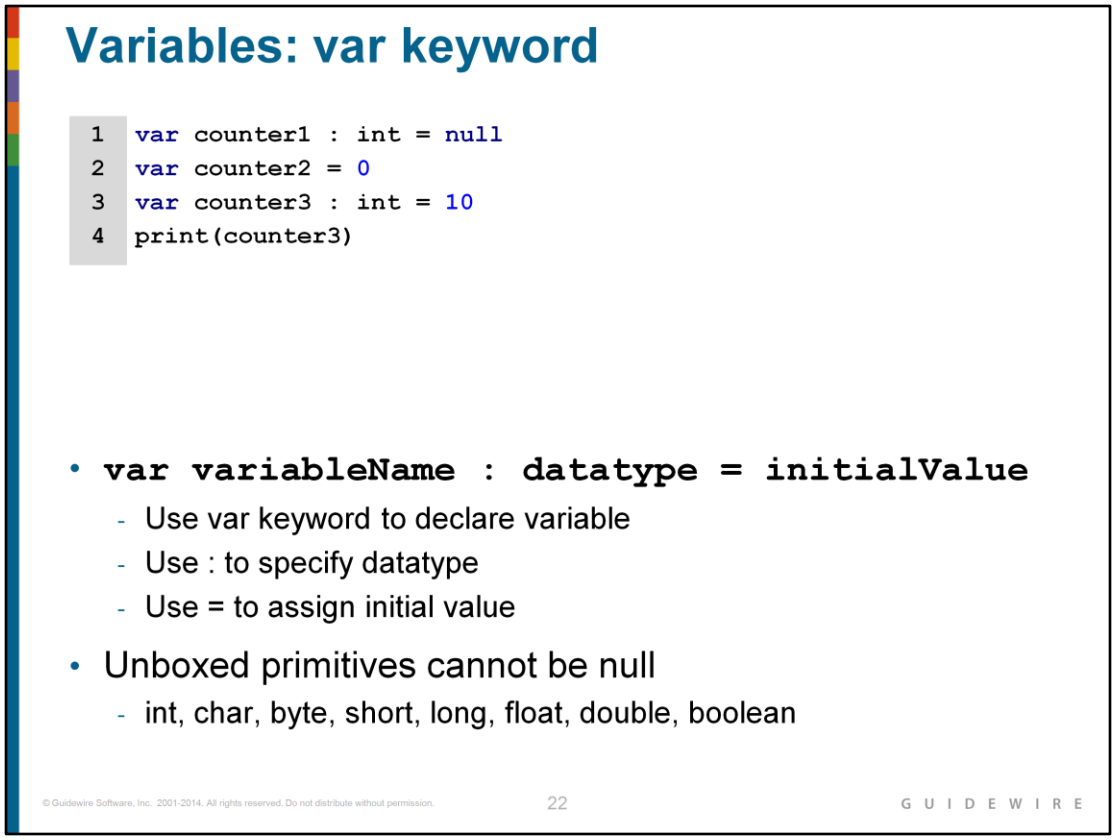
To run in a debug process, you must first start the server in debug mode so that you can execute the code in a debug process. To debug the server, select Main menu → Run → Debug 'Server' or Main menu → Run → Debug… → Server. In the Debug tools window, confirm that the application is running and is ready (***** AppName ready *****). Then, open Gosu Scratchpad using Main menu → Tools → Gosu Scratchpad or with the keystroke ALT+SHIFT+S. In Gosu Scratchpad toolbar, click Run in Debug Process.

You can edit the Gosu Scratchpad program configurations to improve run times in certain cases. To edit the configurations of Gosu Scratchpad, select Main menu → Run → Edit Configurations… . In the Run/Debug Configurations dialog, in the navigation pane, select Gosu Program and then Gosu Scratchpad.gsp. In the Before launch pane, remove Make. Next, in the navigation pane, select Default, then Gosu Program, and then Gosu Scratchpad.gsp. In the Before launch pane, remove Make. Make refers to Make Project, a process in which Guidewire Studio makes the project according to the Project settings.

If your Gosu Scratchpad code references new or changed types (entities, classes, typekeys), removing Make can result in run-time compiler errors. In addition, removing Make from other project settings can result in Studio caching issues. To resolve cache corruption problems, invalidate Studio's caches and restart studio, a process which can take a long time as it involves rebuilding and indexing the project.

Lesson outline

- Gosu overview
- Gosu Scratchpad
- Gosu statements
- Gosu objects
- Gosu subtypes



A variable stores data, object, or collection of objects temporarily in memory. Each variable within has a name. You create a variable using the var keyword. When you declare a variable, you specify its type either directly or indirectly. To explicitly define the variable type, use a colon after the variable name followed by a type name. If a type is specified for a variable with a colon, the variable is considered strongly typed (line 1). If a variable is initialized with a value, but no type is specified, the variable is strongly typed to the type of the value (line 2). Gosu uses the standard programming assignment operator (=) to assign the value on the right-hand side of the statement to the item on the left-hand side of the statement. Line 3 is an example of how to both declare a variable with a type and initial value.

In the slide example, counter1 is a variable of the type int. The type int is a Gosu primitive type. In Gosu, variables declared of a primitive type be null. Behind the scenes, the Gosu Guidewire Platform compiler will coerce the primitive to not throw a Null Pointer Exception. The result is that counter1 is not null, but rather, equal to 0! However, if you execute this code in Gosu Scratchpad using the Community Release compiler, counter1 will throw a Null Pointer Exception.

To execute the code in the slide example, you must Run in a Debug process. To debug the server, select Main menu → Run → Debug 'Server' or Main menu → Run → Debug… → Server. In the Debug tools window, confirm that the application is running and is ready (***** AppName ready *****). Then, open Gosu Scratchpad using Main menu → Tools → Gosu Scratchpad or with the keystroke ALT+SHIFT+S. In Gosu Scratchpad toolbar, click Run in Debug Process.

Gosu primitive types

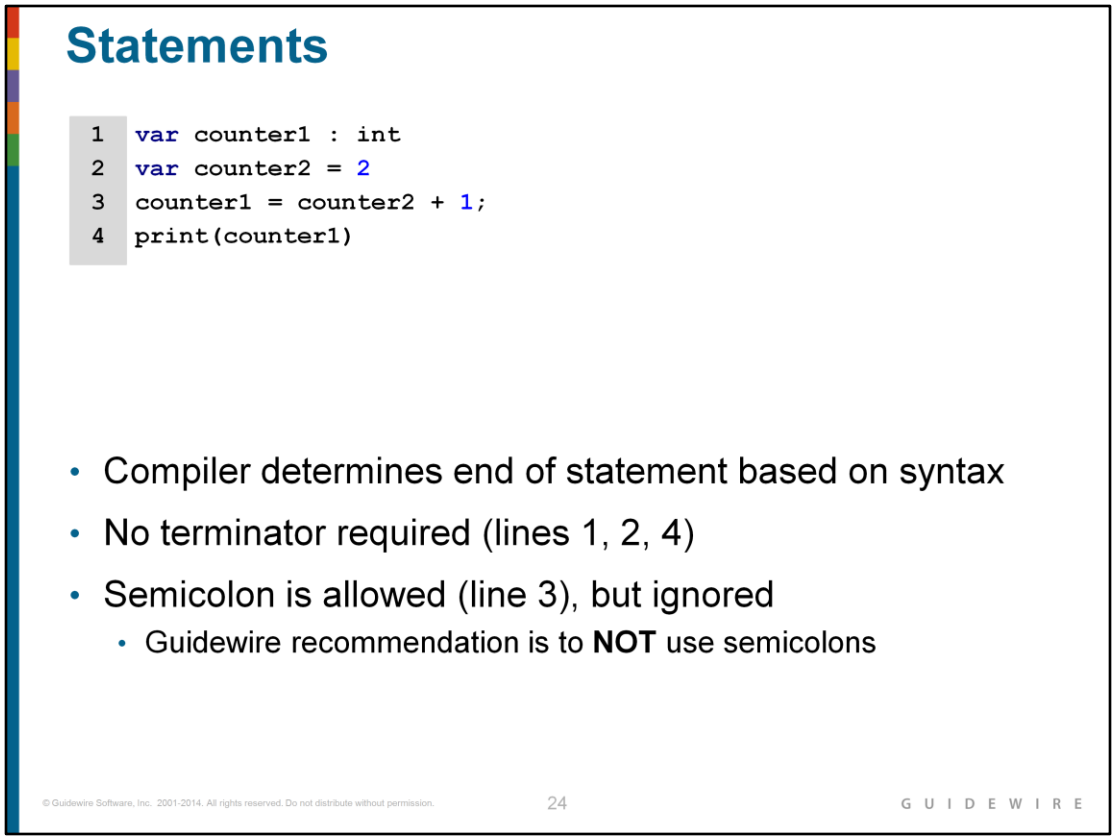
- Exist for compatibility with the Java Language
- Primitive types cannot hold the **null** value
  - Null is special value that means an empty object
- Primitive types coerced to non-primitive versions or back again in Gosu
  - Gosu type conversion apply for **null** values
  - Example
    - null java.lang.Integer converts to 0 int

Gosu supports the following primitive types: int, char, byte, short, long, float, double, boolean, and the special value that means an empty object value: null. Additionally, every Gosu primitive type (other than the special value null) has an equivalent object type defined in Java.

For example, for int there is the java.lang.Integer type that descends from the Object class. The category of object types that represent the equivalent of primitive types are called boxed primitive types. In contrast, Gosu primitive types are called unboxed primitives. In Gosu, primitive types such as int and boolean exist primarily for compatibility with the Java language. Gosu uses Java primitive types to support extending Java classes and implementing Java interfaces. From a Gosu language perspective, primitives are different only in subtle ways from object-based types such as Integer and Boolean. Primitive types can be automatically coerced (converted) to non-primitive versions or back again by the Gosu language in almost all cases.

From Gosu you can access the Java object versions (non-primitives) of the Java primitive types. For example, java.lang.Boolean is an object type that wraps the behavior of the boolean primitive (hence the term boxing). Primitive types do not perform better in terms of performance or space compared to their object versions. In both Gosu and Java, the language primitive types like int and boolean work differently from objects (descendants of the root Object class). For example, you can add objects to a collection, but not primitives.

There are some subtleties about a running Guidewire application applies Gosu-to-Java-to-Gosu coercions. When executing Gosu code in a Guidewire application context, coercions are automatically applied, for example, in Gosu Rules. But, if you execute Gosu code out of application context, for example, in Gosu Scratchpad that is NOT running in a debug process, no coercion takes place because Scratchpad is using the Gosu Community Release compiler.



In some programming languages, statements must be explicitly terminated. The semi-colon is one of the most common statement terminators, though some languages use other symbols and some use a carriage return.

Semicolons can be used in Gosu to terminate statements. A semicolon is not required, however, because the compiler can parse statements without them. Given that they are not required, Guidewire recommends that you not use semicolons. Guidewire also recommends using carriage returns and white space to make it clear to others reading the code where a given statement ends. The semicolon is allowed (but ignored) as a line terminator.

You can execute the code in the slide example in either a Scratchpad context or Debug Server process. To execute in Scratchpad, select Main menu → Run → Run 'Gosu Scratchpad.gsp'. You can also select Debug 'Gosu Scratchpad.gsp' from the main menu of Studio: Main menu → Run → Debug 'Gosu Scratchpad.gsp'. To debug the server, select Main menu → Run → Debug 'Server' or Main menu → Run → Debug… → Server. In the Debug tools window, confirm that the application is running and is ready (***** AppName ready *****). Then, open Gosu Scratchpad using Main menu → Tools → Gosu Scratchpad or with the keystroke ALT+SHIFT+S. In Gosu Scratchpad toolbar, click Run in Debug Process.

Comments

```
1  //  variables
2  var counter1 : int // defaults to 0 because is an int
3  var counter2 = 0
4  /*
5  Begin Computation
6  */
7  counter1 = counter2 + 1
8  print(counter1)
```

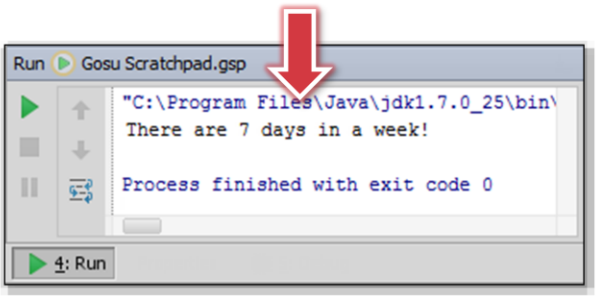
- A comment is a set of characters that the compiler does not attempt to compile
  - End of line comments //
  - Single-line comment with **CTRL+/** (line 1)
  - Multiple-line comment with **CTRL+SHIFT+/** (lines 3,4,5)
- Toggle single-line and multiple-line comments with keystrokes

A comment is a set of characters that the compiler does not attempt to compile. Comments are used by the programmer to document some fact about the code, such as when it was written, what its purpose is, and so on. Comments are also used to make code more readable.

A "//" comments out the remainder of the line until a carriage return is encountered. A "/*" comments out all text until a "*/" is encountered.
Gosu uses the same comment syntax as Java.

Concatenation operator

```
1  var count : int = 7
2  var out : String = "Current Count: "
3  out += "There are " + count + " days in a week!"
4  print(out)
```



- + concatenates two or more values into a single string
- += concatenates variable with right side expression
- Syntax: **`variable += value + value`**

Line 3 illustrates the syntax for concatenation in a variable assignment (+=). Rather than writing out=out + "my string", you can write out += "my string".

You can execute the code in the slide example in either a Scratchpad context or Debug Server process. To execute in Scratchpad, select Main menu → Run → Run 'Gosu Scratchpad.gsp'. You can also select Debug 'Gosu Scratchpad.gsp' from the main menu of Studio: Main menu → Run → Debug 'Gosu Scratchpad.gsp'. To debug the server, select Main menu → Run → Debug 'Server' or Main menu → Run → Debug… → Server. In the Debug tools window, confirm that the application is running and is ready (***** AppName ready *****). Then, open Gosu Scratchpad using Main menu → Tools → Gosu Scratchpad or with the keystroke ALT+SHIFT+S. In Gosu Scratchpad toolbar, click Run in Debug Process.

Statements: if-else

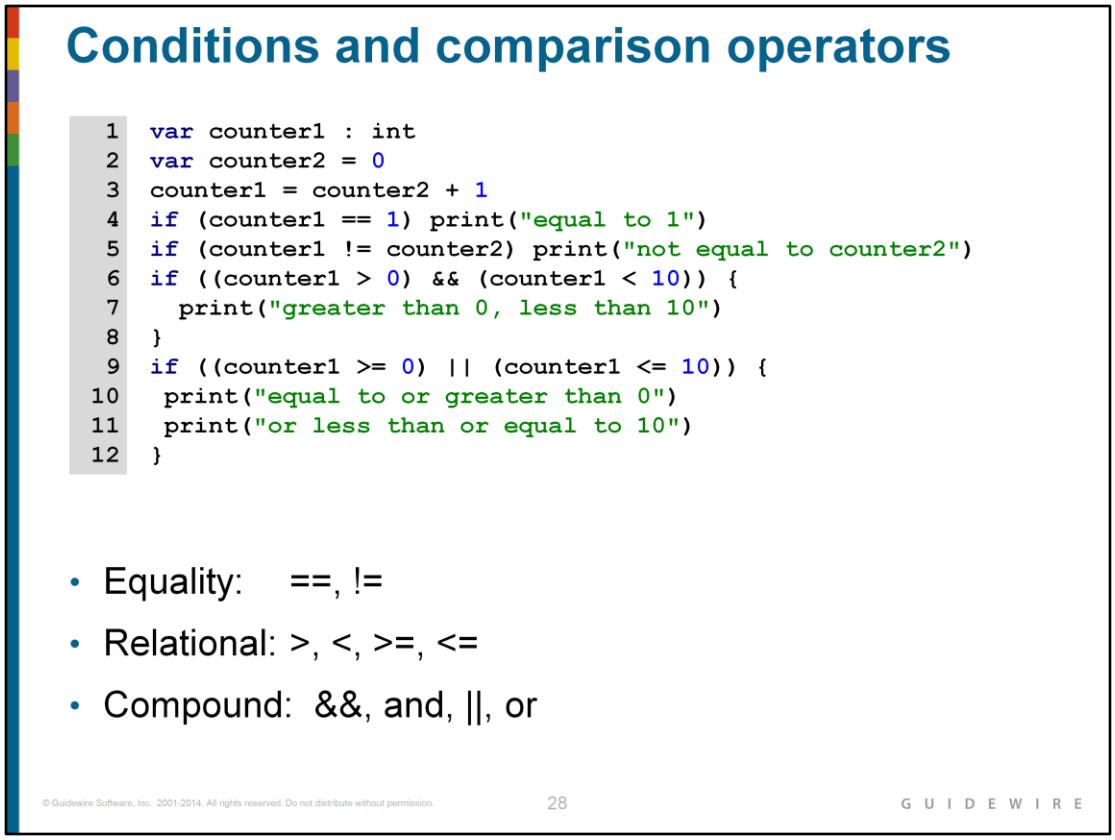
```
1  var a : boolean = true
2  var b = false
3  var c = false
4  if (a == b) {
5    print(a)
6  }
7  else if (b == c) {
8    print (c)
9  }
10 else {
11   print(null)
12 }
```

- Evaluates condition to be true or false and is followed by a single statement or block of statements ({})
- Syntax:

```
if (condition)
    statement_or_{block}
else
   statement_or_{block}
```


The most commonly used statement block within the Gosu language is the if block. The if block uses a multi-part construction. The else block is optional. If there is more than one else, you can use else if.

You can execute the code in the slide example in either a Scratchpad context or Debug Server process. To execute in Scratchpad, select Main menu → Run → Run 'Gosu Scratchpad.gsp'. You can also select Debug 'Gosu Scratchpad.gsp' from the main menu of Studio: Main menu → Run → Debug 'Gosu Scratchpad.gsp'. To debug the server, select Main menu → Run → Debug 'Server' or Main menu → Run → Debug… → Server. In the Debug tools window, confirm that the application is running and is ready (***** AppName ready *****). Then, open Gosu Scratchpad using Main menu → Tools → Gosu Scratchpad or with the keystroke ALT+SHIFT+S. In Gosu Scratchpad toolbar, click Run in Debug Process.



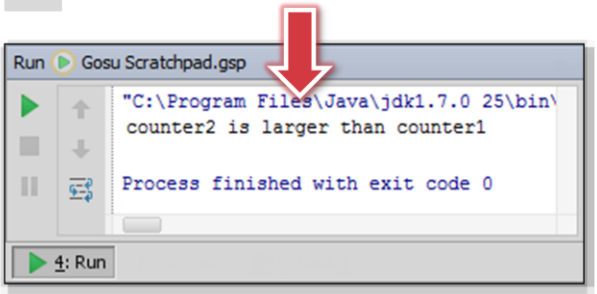
The AND (&&) and OR (||) operators can also be entered as "and" and "or" (using lower-case letters). As with most other scripting languages, in Gosu the "is equal to" comparison operator uses two equals signs ("=="), while the assignment operator uses one ("="). The != operator tests for relational inequality. The operands can be of any compatible types. The result is always Boolean. For example:

```
var count = 3       // Declare a new variable named "count" and
                    // assign it the number 3.
count = 4           // Now assign it the number 4
if (count == 4 )...// The if expression is true because "count"
                    // contains the number 4.
if (3 == 4)...      // The if expression is false because 3 does
                    // not equal 4.
if (3 != 4)...      // The if expression is true because 3 does
                    // not equal 4.
3 = 4               // This statement will not compile. You cannot
                    // assign a value (such as 4) to a number
                    // (such as 3).
```

In the Gosu language, the == operator automatically calls *object.*equals() for comparison if you use it with reference types. In most cases, this is what you want for reference types. However, there are some cases in which you want to use identity reference, not simply comparing the values using the underlying object.equals() comparison. In other words, sometimes you want to know if two objects literally reference the same in-memory object. You can use the Gosu operator === (three equals signs) to compare object equality. This always compares whether both references point to the same in-memory object.

Ternary operator

```
1  var counter1 : int = 100
2  var counter2 : int = 200
3  print((counter1 > counter2)
4    ? "counter1 is larger than counter2"
5    : "counter2 is larger than counter1")
```



- Similar to if-else, but can only return expressions and cannot execute statements
- Often widget properties will user ternary operator
  - If-else statements are cumbersome

Parentheses are not required around the condition of a ternary operator. They are often added for the sake of clarity or to establish an order of precedence, however.

Guidewire static method libraries

```
1  var currentDate = gw.api.util.DateUtil.currentDate()
2  var randomNumber = gw.api.util.Math.random(1000)
3  var output = gw.api.util.StringUtil.formatDate(currentDate,
4                                                 "YYYY-MM-DD")
5  if (randomNumber > 500) {
6    print (currentDate + " " + randomNumber + " " + output )
7  }
```

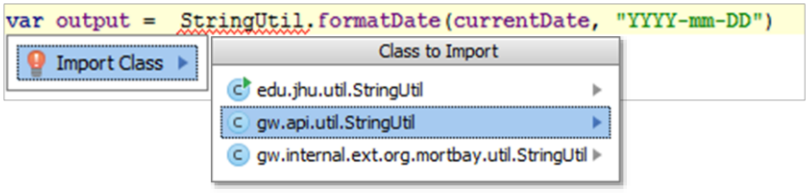
- Gosu supports static members on a type
  - Variables, functions, and property declarations
- Reference class and static members

A static member means that the member exists only on the single type itself, not on *instances* of the type. Access static members on Java types just as you would native Gosu types. For Gosu code that accesses static members, you must qualify the class that declares the static member.

Import packages: uses operator

```
1  uses gw.api.util.DateUtil
2  uses gw.api.util.Math
3  uses gw.api.util.StringUtil
4  var currentDate = DateUtil.currentDate()
5  var randomNumber = Math.random(1000)
6  var output = StringUtil.formatDate(currentDate, "YYYY-MM-DD")
7  if (randomNumber > 500) {
8    print (currentDate + " " + randomNumber + " " + output )
9  }
```

- Imports fully qualified name and package
  - Not a static import (like Java), must reference class in code
  - Asterisk (*) to import hierarchy
- Studio will suggest class while typing class name



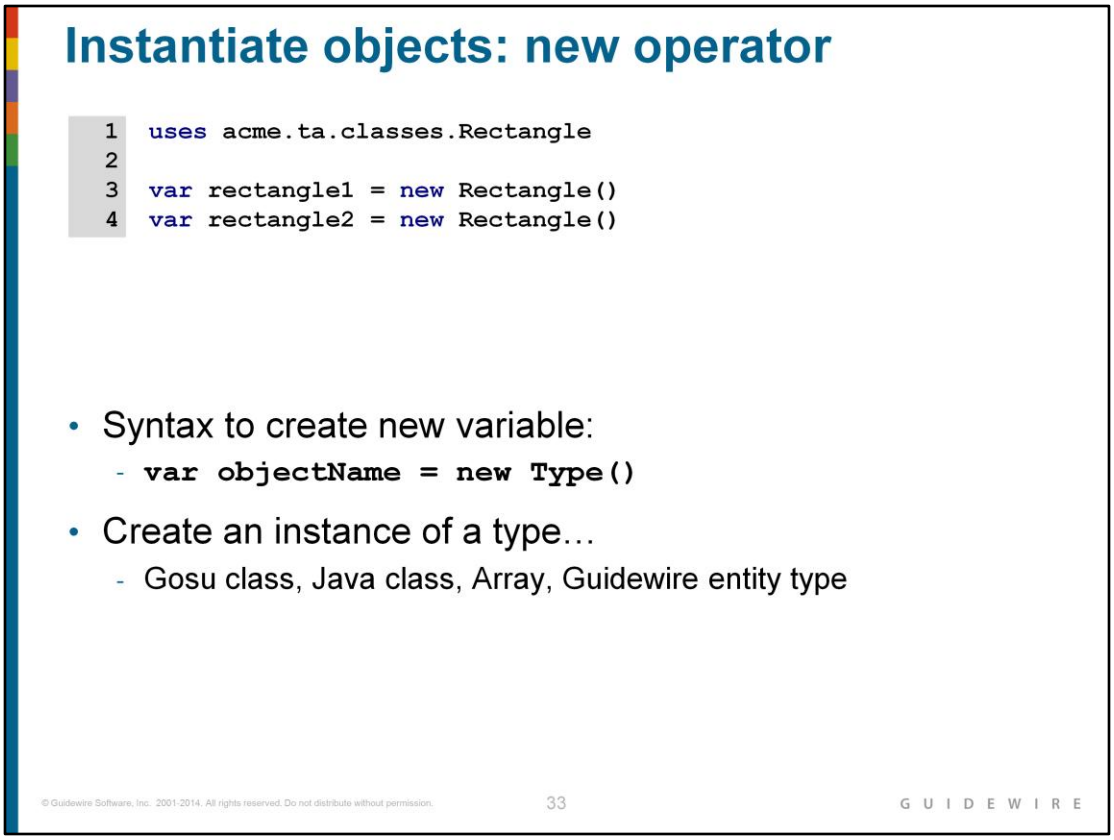
To use types and namespaces in Gosu scripts without fully qualifying the full class name including the package, use the Gosu **uses** operator. By convention, put uses imports at the beginning of the file or script. The uses operator behaves in a similar fashion to the Java language's import command, however, explicit types always have precedence over wildcard namespace references. Namespaces can be specified with an asterisk (*) character to indicate a hierarchy.

While the **uses** operator is technically an unary operator in that it takes a single operand, the functionality it provides is only useful with a second statement. In other words, the only purpose of using a uses expression is to simplify other lines of code in which you can omit the fully-qualified type name.

You can type in the class name and method and Studio will try to resolve the fully qualified name for you if the package or namespace has not been already declared. Use ALT+ENTER over the red squiggles to see the Import Class options.

Lesson outline

- Gosu overview
- Gosu Scratchpad
- Gosu statements
- Gosu objects
- Gosu subtypes



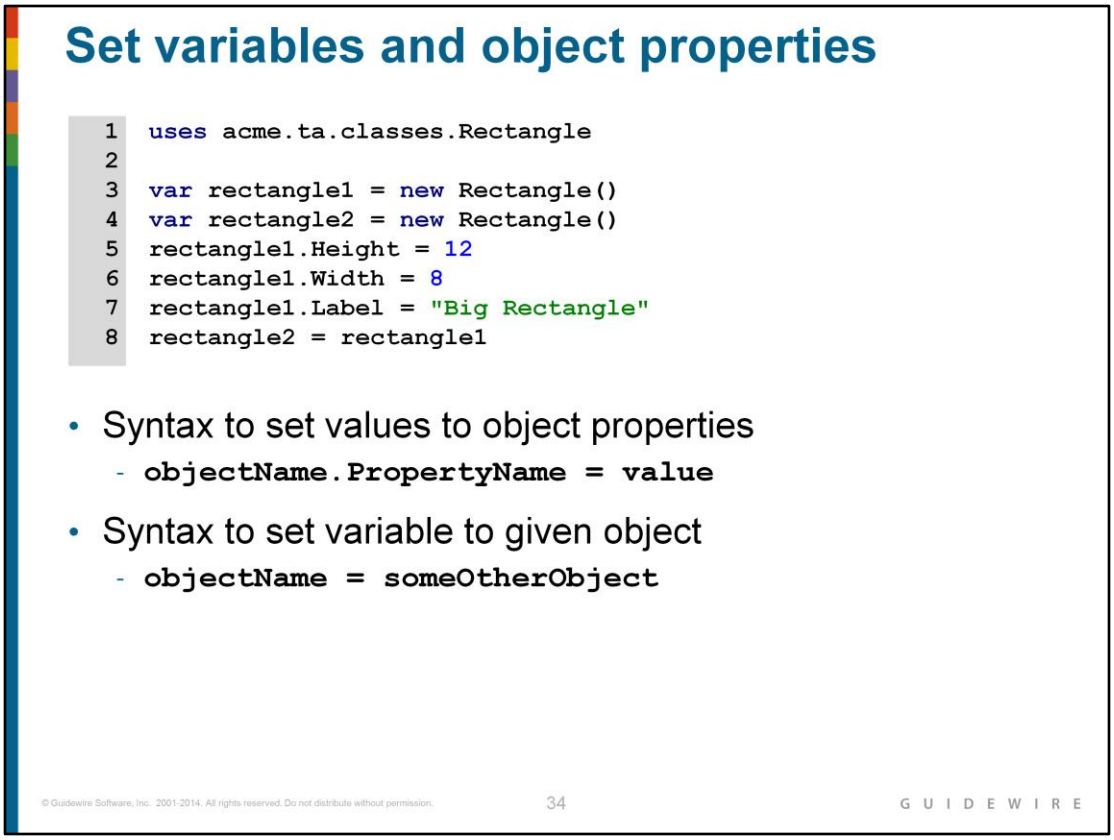
Gosu uses the **new** operator to create an instance of a type. The type can be a Gosu class, a Java class, an array, or a Guidewire entity type.

All Gosu values have a type. Object instances have types. The type of an instance of a class is the class itself.

You can use the new operator with any valid Gosu type, Java type, or an array. At least one constructor (creation function) must be exposed on a type to construct an instance of the type with the new operator.

Although the new operator is often used as an expression, it can also be a statement. For some types, this may not be useful. However, if the constructor for the object triggers code that saves a copy of the new object, the return value from new may be unnecessary. Ignoring the return value and using new as a statement may permit more concise code in some cases. For example, suppose that a constructor for a class that represents a book registers itself with a bookshelf object and saves the new object. Some code might simply create the book object and pass the bookshelf as a constructor argument.

In the slide example, line 1 imports the Rectangle class. Lines 3-4 instantiate two Rectangle objects, rectangle1 and rectangle2.



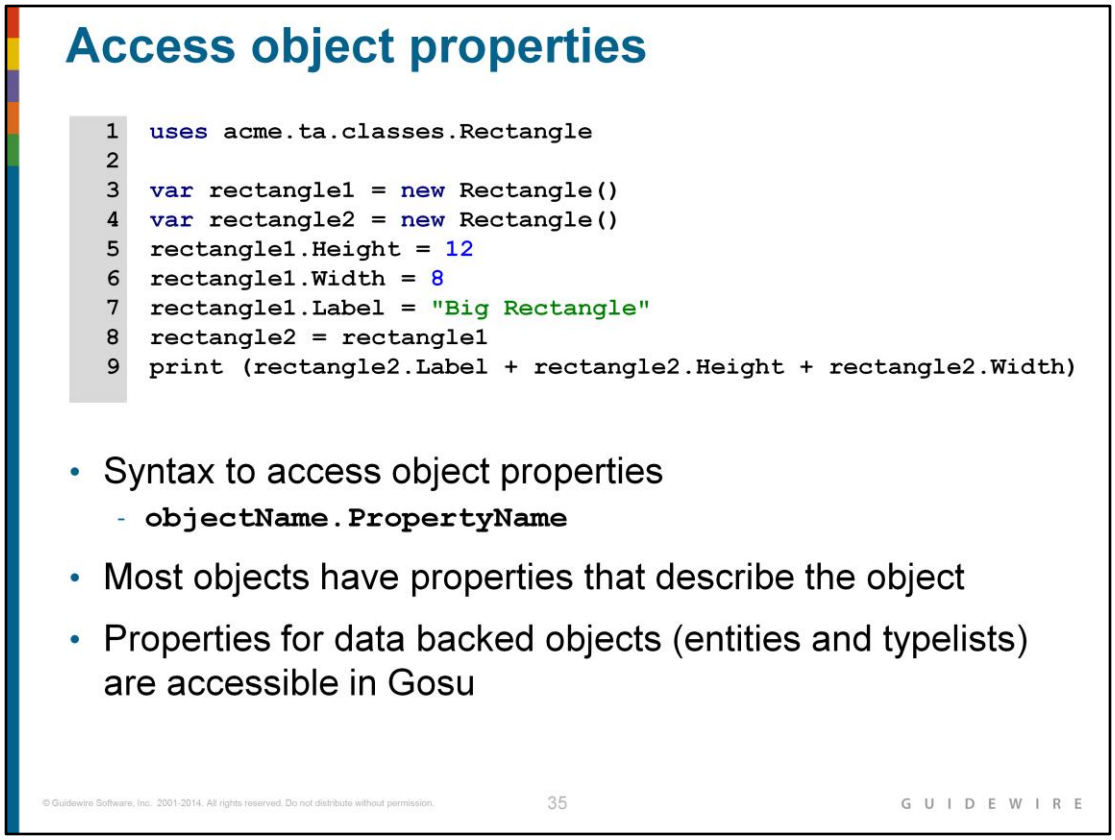
Gosu classes can define properties. A property can be thought of as a value associated with an object. A property has a type and its value can be retrieved or set. Class properties appear to other objects like variables on the class. Use the period symbol (.) to access a property by name.

The class Rectangle describes private variables for the Height, Width, and Label properties. When a value is assigned to an object property, the data is stored in a private variable. Code that accesses the property data does not access the private instance variable directly. Any code that wants to set the value can do so using the objectName.PropertyName = value syntax.

You can also use an object initializer syntax for setting object properties in the object constructor. An example of a simple object initializer is:
var rectangle = new Rectangle() {:Height = 12, :Width =8, :Label = "My Rectangle"}

Object initializers comprise one or more property initializer expressions, separated by commas, and enclosed by curly braces. A property initializer expression is the following in order: a colon (:), a property name, an equals symbol (=), and a value or any expression that results in a value.

In the slide example, lines 5-7 set values for the properties of rectangle1. Line 8 sets rectangle2 to equal the characteristics of rectangle1.



Use the period symbol (.) to access a property by name.

The class Rectangle describes private variables for the Height, Width, and Label properties. When a value is assigned to an object property, the data is stored in a private variable. Code that accesses the property data does not access the private instance variable directly. Any code that wants to access the value can do so using the objectName.PropertyName.

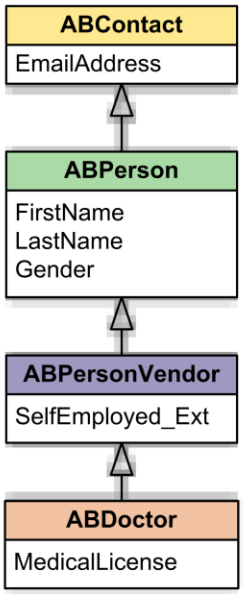
In the slide example, line 9 outputs a string of rectangle2 properties to the console.

Lesson outline

- Gosu overview
- Gosu Scratchpad
- Gosu statements
- Gosu objects
- Gosu subtypes

Subtyped entities

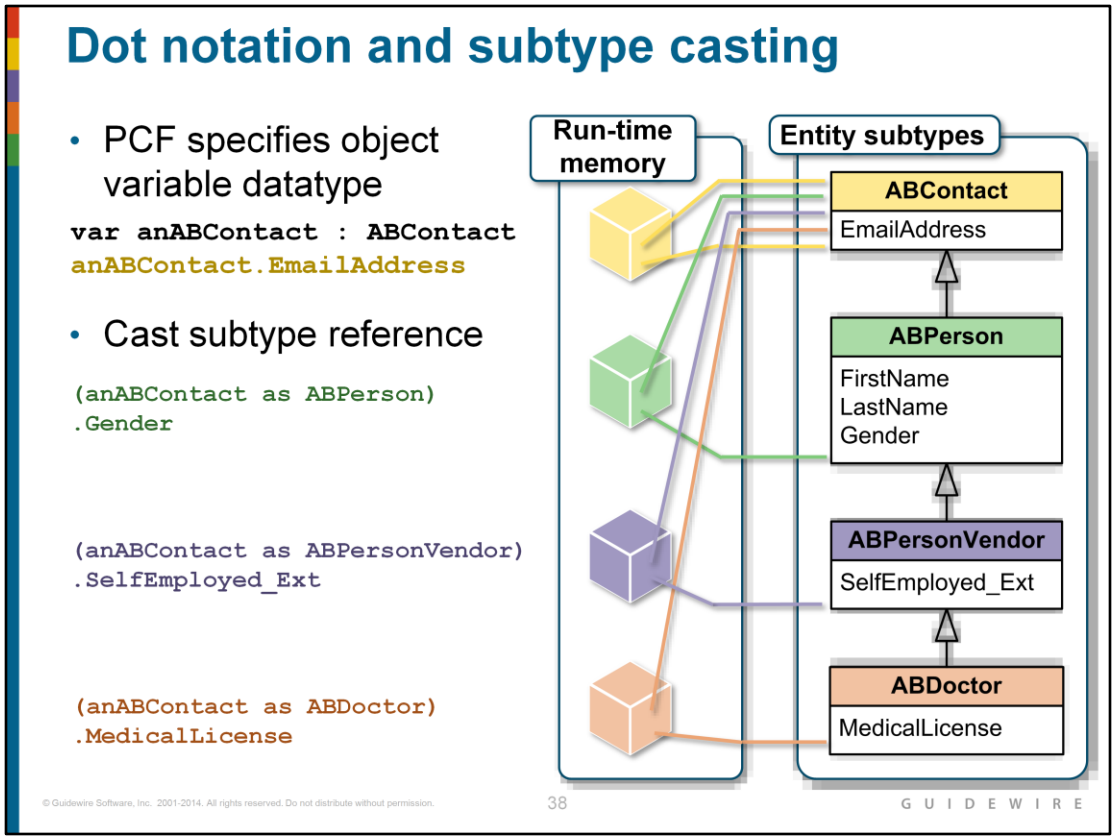
- Guidewire entities may have subtypes
- Each subtype inherits the properties and methods of all their supertypes
- Subtypes typically have their own properties and/or methods



A subtype is an entity that is declared as a child of another "parent" entity. The child entity inherits all properties and methods of the parent. For example, in the slide example, ABPerson is a subtype of ABContact. An object of type ABPerson has FirstName and LastName properties which are declared explicitly in the ABPerson entity. An object of type ABPerson also inherits the properties of the supertype which is ABContact. EmailAddress is a property that ABPerson inherits from ABContact.

Subtypes are useful when you have a hierarchical collection of objects where the same sort of information is needed for sets of objects. For example, all contacts have assigned users and email addresses. All person contacts (but not companies or places) have first names and last names. By creating ABPerson as a subtype of ABContact, you do not need to re-declare the fields shared by ABPerson, ABPlace, and ABCompany.

Create a new subtype only if you need to treat one set of entities differently from another. As much as possible, limit the number of subtypes for representing an entity. The main reason to limit creation of new subtypes is that the more you specialize the subtype hierarchy, the more restrictive and the less flexible your model becomes. For example, there are subtypes for AutoRepairShop and AutoTowingAgcy. If you work with an auto repair shop that also does towing, you cannot create a single contact that does both. The reason is that you cannot create a subtype that inherits the fields of two or more entities directly. To create the subtype you want, you can select one entity or the other as the subtype. Then, add the fields that are missing from the other entity to you subtype.



When a Guidewire application retrieves data from the database, it stores that data as an object in runtime memory. It works with the in-memory information until the data needs to be committed or re-retrieved from the database.

When an object is created, all of the information about the object is copied into memory. If the object is subtyped, then data for all the fields at every relevant subtype level is retrieved. For example, if an object is an ABDoctor, then information is copied over for fields at the ABContact, ABPerson, ABPersonVendor, and ABDoctor levels. Fields associated with ABPlace, ABCompany, or ABAttorney are irrelevant and ignored.

When an object is referenced, the server uses the datatype of the reference to understand the structure of the information in memory. For example, assume that there is an object named "anABContact" that stores information about an ABDoctor. If there is a reference to this object with a datatype of "ABDoctor", then the server knows that the object will have ABContact fields, ABPerson fields, ABPersonVendor fields, and ABDoctor fields. If there is a reference to this object with a datatype of "ABPerson", then the server knows that the object will have ABContact fields and ABPerson fields, but it will assume that the object has no fields beyond the ABPerson fields. There may be additional fields with information at the ABPersonVendor and ABDoctor levels, but the server will be unaware of them.

In some cases, a method or user interface container receives a reference to an object, and the reference uses a "high-level" datatype (such as ABContact). This reference cannot be used as-is to access fields below the specified level because the server assumes those fields don't exist. There is a programming technique known as casting in which you explicitly state the datatype for a subtyped object. This is done using the syntax "as <datatype>", where <datatype> is the datatype that you want to explicitly identify. When you cast a reference, you are providing the server with more information about the structure of the object. A casted reference can access fields that an uncasted reference cannot.

Direct reference of subtype properties

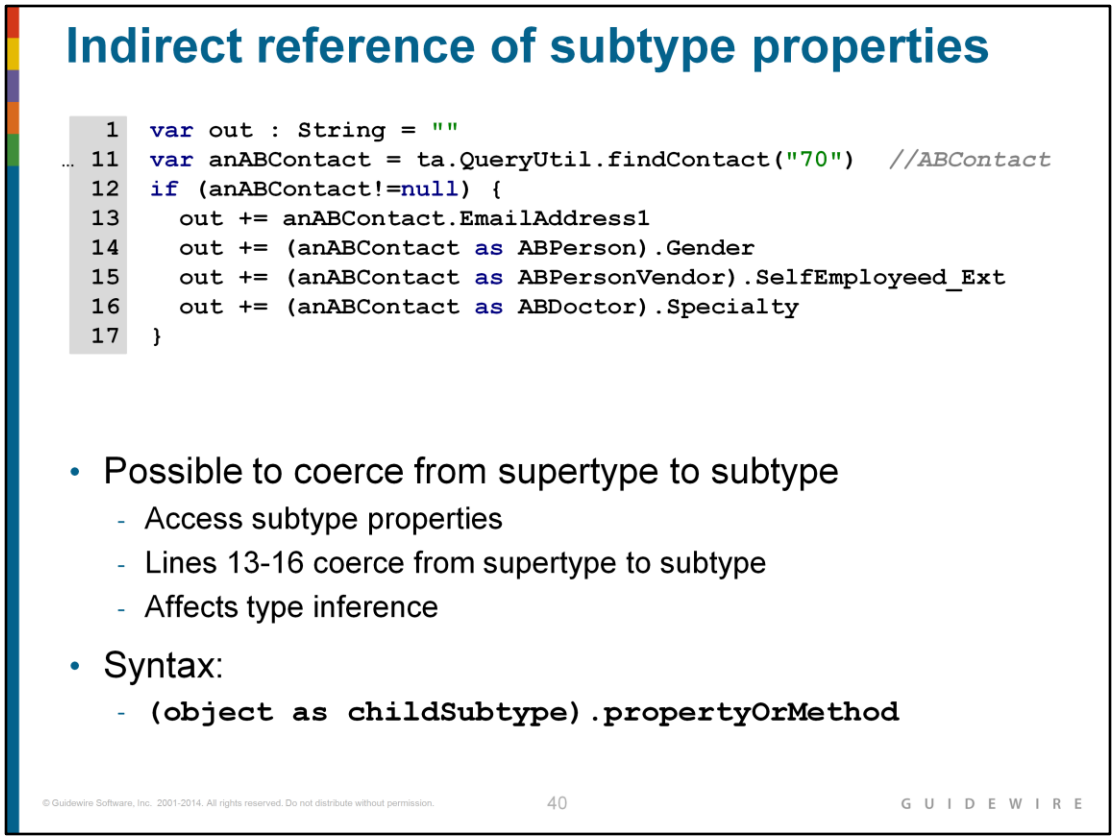
```
1  var out : String = ""
2  var anABDoctor = ta.QueryUtil.findDoctor("70") //ABDoctor
3
4  if (anABDoctor!=null) {
5    out += anABDoctor.EmailAddress1  //ABContact defines
6    out += anABDoctor.Gender  //ABPerson defines
7    out += anABDoctor.SelfEmployeed_Ext  //ABPersonVendor defines
8    out += anABDoctor.Specialty  //ABDoctor defines
9  }
10
11 var anABContact = ta.QueryUtil.findContact("70")  //ABContact
12 if (anABContact!=null) {
13   out += anABContact.EmailAddress1  //ABContact defines
14   out += anABContact.Gender  //ABPerson defines
15   out += anABContact.SelfEmployeed_Ext  //ABPersonVendor defines
16   out += anABContact.Specialty  //ABDoctor defines
17 }
```

- Only can reference properties at subtype and above
  - Lines 5-8 execute
  - Lines 13-16 won't execute: *No property descriptor found*

In the slide example, the Gosu code retrieves two objects: anABDoctor of the type ABDoctor and anABContact of the type ABContact. ABContact is the supertype of ABPerson; ABPerson is the supertype of ABPersonVendor; and ABPersonVendor is the supertype of ABDoctor. Expressed another way, ABDoctor is the subtype of ABPersonVendor; ABPersonVendor is the subtype of ABPerson; ABPerson is the subtype of ABContact.

Lines 5-8 illustrate that all the supertype properties of anABDoctor are accessible.
Lines 14-16 illustrate that you cannot access the subtype properties when a supertype.

To execute the code in the slide example, you must Run in a Debug process. To debug the server, select Main menu → Run → Debug 'Server' or Main menu → Run → Debug… → Server. Then, open Gosu Scratchpad using Main menu → Tools → Gosu Scratchpad or with the keystroke ALT+SHIFT+S. In Gosu Scratchpad toolbar, click Run in Debug Process.



Lines 14-16 illustrate that you can coerce to a subtype and access the subtype properties.

Gosu uses the "expression as TYPE" construction to cast an expression to a specific type which is known as coercion. The expression must be compatible with the type. If you try to cast an expression to an inappropriate type, Gosu throws an exception.

If an object has a compile-time type that is higher in the type hierarchy (it is a supertype) than you need, coerce it to the appropriate specific type. Before you can access properties or methods on the object defined in a subtype from a supertype, you must coerce the type to the subtype. Gosu provides automatic downcasting to simplify your code in if statements and similar structures.

Subtype property

```
1  var out : String = ""
2  var anABContact = ta.QueryUtil.findContact("70")
3  out += anABContact.EmailAddress1
4  / * Begin condition statements for subtypes */
5  if (anABContact.Subtype==typekey.ABContact.TC_ABPERSON) {
6    out += (anABContact as ABPerson).Gender
7  }
8  if (anABContact.Subtype==typekey.ABContact.TC_ABPERSONVENDOR) {
9    out += (anABContact as ABPersonVendor).SelfEmployeed_Ext
10 }
11 if (anABContact.Subtype==typekey.ABContact.TC_ABDOCTOR) {
12   out += (anABContact as ABDoctor).Specialty
13 }
14 / * End condition statements for subtypes */
15 print(out)
```

- An entity's Subtype property is a typekey field of the entity subtypes

The Subtype property identifies a particular subtype within a supertype table. Each subtype of a supertype has its own unique subtype value. In the slide example, various condition statements contain expressions to determine if the anABContact object is a specify subtype. If the anABContact object is of the type ABDoctor, then all condition statements will be true.

Line 1 declares the out variable of type String with an empty value.
Line 2 returns a contact from a query. The findContact() method requires a string identifier as a parameter.

Line 3 concatenates the object's EmaillAddress1 property with the out string. All contacts are of the type ABContact, the supertype.

Lines 5-7 test to see if the anABContact object is of the subtype ABPerson. If anABContact is an ABPerson subtype, then the object is cast to ABPerson and the out string concatenates the object's Gender property.

Lines 8-10 test to see if the anABContact object is of the subtype ABPersonVendor. If anABContact is an ABPersonVendor subtype, then the object is cast to ABPersonVendor and the out string concatenates the object's SelfEmployeed_Ext property.

Lines 11-13 test to see if the anABContact object is of the subtype ABDoctor. If anABContact is an ABDoctor subtype, then the object is cast to ABDoctor and the out string concatenates the object's Specialty property.

Type checking: typeis operator

```
1  var out : String = ""
2  var anABContact = ta.QueryUtil.findContact("70")
3  out += anABContact.EmailAddress1
4  / * Begin condition statements for subtype with downcasting */
5  if (anABContact typeis ABPerson) {
6    out += anABContact.Gender
7  }
8  if (anABContact typeis ABPersonVendor) {
9    out += anABContact.SelfEmployeed_Ext
10 }
11 if (anABContact typeis ABDoctor) {
12   out += anABContact.Specialty
13 }
14 / * End condition statements for subtypes with downcasting*/
15 print(out)
```

- Compare an expression's type with a specified type
- Automatic downcasting when in expression of if block
  - No need to cast the object explicitly

Gosu uses the typeis operator to compare an expression's type with a specified type. The result is always Boolean. A typeis expression cannot be fully determined at compile time. For example, at run time the expression may evaluate to a more specific subtype than the variable's declared type.

Gosu automatically downcasts an object after a typeis expression if the type is a subtype of a given supertype. In other words, when using the typeis expression within the if block statement, you do not need to explicitly cast an object (as TYPE expressions) to that subtype. The typeis expression for a subtype implicitly considers that variable's type to be the subtype within the scope of the "if block" of code. If you case the object explicitly, and Gosu provides a warning.

Often, you need to check an object against a type or its subtypes, not just a single type. In such cases, it is better to use typeis instead of typeof. The typeof keyword returns the exact type. If you test this value with simple equality with another type, it returns false if the object is a subtype.

Lesson objectives review

- You should now be able to:
  - Describe the ways that Guidewire applications use Gosu
  - Write Gosu using basic Gosu syntax
  - Describe Studio features that aid in the writing of Gosu

Review questions

1. Name three Gosu primitives.
2. When does Scratchpad use…
   a) the Gosu Guidewire Community Release compiler?
   b) the Gosu Guidewire Platform compiler?
3. For a class named DateUtil, Gosu Scratchpad suggests four different packages. How can you specify which to use throughout your Gosu Scratchpad code?
4. Your code returns various subtype objects. How can you test to see if a given object has a specific property without throwing an exception.

Answers
1) Any of the following: int, char, byte, short, long, float, double, boolean.
2a) When Gosu Scratchpad is running in its own process (debug or run), then Scratchpad uses the Gosu Guidewire Community Release compiler.
2b) When Gosu Scratchpad is running in a Guidewire application process (Debug 'Server'), then Scratchpad uses the Gosu Guidewire Platform compiler.
3) Import the fully qualified class name (package) with the uses operator at the top of the file.
4) Use an if-else condition statement with the typeis operator to downcast the object without throwing an exception.

Notices