JAVA

programming language high level

Author

PIERMATTEO BARAMBANI

Created by WikiCourseBuilder

1. C Sharp (programming language)

1.1.

C# is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft within its .NET initiative and later approved as a standard by Ecma (ECMA-334) and ISO (ISO/IEC 23270:2006). C# is one of the programming languages designed for the Common Language Infrastructure.

C# is a general-purpose, object-oriented programming language. Its development team is led by Anders Hejlsberg. The most recent version is C# 7.2, which was released in 2017 along with Visual Studio 2017 version 15.5.

1.2. Design goals

The ECMA standard lists these design goals for C#:

1. The language is intended to be a simple, modern, general-purpose, object-oriented programming language.
2. The language, and implementations thereof, should provide support for software engineering principles such as strong type checking, array bounds checking, detection of attempts to use uninitialized variables, and automatic garbage collection. Software robustness, durability, and programmer productivity are important.
3. The language is intended for use in developing software components suitable for deployment in distributed environments.
4. Portability is very important for source code and programmers, especially those already familiar with C and C++.
5. Support for internationalization is very important.
6. C# is intended to be suitable for writing applications for both hosted and embedded systems, ranging from the very large that use sophisticated operating systems, down to the very small having dedicated functions.
7. Although C# applications are intended to be economical with regard to memory and processing power requirements, the language was not intended to compete directly on performance and size with C or assembly language.

1.3. History

During the development of the .NET Framework, the class libraries were originally written using a managed code compiler system called Simple Managed C (SMC). In January 1999, Anders Hejlsberg formed a team to build a new language at the time called Cool, which stood for "C-like Object Oriented Language". Microsoft had considered keeping the name "Cool" as the final name of the language, but chose not to do so for trademark reasons. By the time the .NET project was publicly announced at the July 2000 Professional Developers Conference, the language had been renamed C#, and the class libraries and ASP.NET runtime had been ported to C#.

Hejlsberg is C#'s principal designer and lead architect at Microsoft, and was previously involved with the design of Turbo Pascal, Embarcadero Delphi (formerly CodeGear Delphi, Inprise Delphi and Borland Delphi), and Visual J++. In interviews and technical papers he has stated that flaws in most major programming languages (e.g. C++, Java, Delphi, and Smalltalk) drove the fundamentals of the Common Language Runtime (CLR), which, in turn, drove the design of the C# language itself.

James Gosling, who created the Java programming language in 1994, and Bill Joy, a co-founder of Sun Microsystems, the originator of Java, called C# an "imitation" of Java; Gosling further said that " sort of Java with reliability, productivity and security deleted." Klaus Kreft and Angelika Langer (authors of a C++ streams book) stated in a blog post that "Java and C# are almost identical programming languages. Boring repetition that lacks innovation," "Hardly anybody will claim that Java or C# are revolutionary programming languages that changed the way we write programs," and "C# borrowed a lot from Java - and vice versa. Now that C# supports boxing and unboxing, we'll have a very similar feature in Java."
In July 2000, Hejlsberg said that C# is "not a Java clone" and is "much closer to C++" in its design.

Since the release of C# 2.0 in November 2005, the C# and Java languages have evolved on increasingly divergent trajectories, becoming somewhat less similar. One of the first major departures came with the addition of generics to both languages, with vastly different implementations. C# makes use of reification to provide "first-class" generic objects that can be used like any other class, with code generation performed at class-load time.
Furthermore, C# has added several major features to accommodate functional-style programming, culminating in the LINQ extensions released with C# 3.0 and its supporting framework of lambda expressions, extension methods, and anonymous types. These features enable C# programmers to use functional programming techniques, such as closures, when it is advantageous to their application. The LINQ extensions and the functional imports help developers reduce the amount of boilerplate code that is included in common tasks like querying a database, parsing an xml file, or searching through a data structure, shifting the emphasis onto the actual program logic to help

improve readability and maintainability.

C# used to have a mascot called Andy (named after Anders Hejlsberg). It was retired on January 29, 2004.

C# was originally submitted to the ISO subcommittee JTC 1/SC 22 for review, under ISO/IEC 23270:2003, was withdrawn and was then approved under ISO/IEC 23270:2006.

C-sharp musical note
The name "C sharp" was inspired by musical notation where a sharp indicates that the written note should be made a semitone higher in pitch.
This is similar to the language name of C++, where "++" indicates that a variable should be incremented by 1. The sharp symbol also resembles a ligature of four "+" symbols (in a two-by-two grid), further implying that the language is an increment of C++.

Due to technical limitations of display (standard fonts, browsers, etc.) and the fact that the sharp symbol () is not present on most keyboard layouts, the number sign () was chosen to approximate the sharp symbol in the written name of the programming language.
This convention is reflected in the ECMA-334 C# Language Specification. However, when it is practical to do so (for example, in advertising or in box art), Microsoft uses the intended musical symbol.

The "sharp" suffix has been used by a number of other .NET languages that are variants of existing languages, including J# (a .NET language also designed by Microsoft that is derived from Java 1.1), A# (from Ada), and the functional programming language F#. The original implementation of Eiffel for .NET was called Eiffel#, a name retired since the full Eiffel language is now supported. The suffix has also been used for libraries, such as Gtk# (a .NET wrapper for GTK+ and other GNOME libraries) and Cocoa# (a wrapper for Cocoa).

1. Generics
2. Partial types
3. Anonymous methods
4. Iterators
5. Nullable types
6. Getter/setter separate accessibility
7. Method group conversions (delegates)
8. Co- and Contra-variance for delegates
9. Static classes
10. Delegate inference

1. Implicitly typed local variables
2. Object and collection initializers
3. Auto-Implemented properties
4. Anonymous types
5. Extension methods
6. Query expressions
7. Lambda expression
8. Expression trees
9. Partial methods

1. Dynamic binding
2. Named and optional arguments
3. Generic co- and contravariance
4. Embedded interop types ("NoPIA")

1. Asynchronous methods
2. Caller info attributes

1. Compiler-as-a-service (Roslyn)
2. Import of static type members into namespace
3. Exception filters
4. Await in catch/finally blocks
5. Auto property initializers
6. Default values for getter-only properties
7. Expression-bodied members
8. Null propagator (null-conditional operator, succinct null checking)
9. String interpolation
10. nameof operator
11. Dictionary initializer

1. Out variables
2. Pattern matching
3. Tuples
4. Deconstruction
5. Local functions

6. Digit separators
7. Binary literals
8. Ref returns and locals
9. Generalized async return types
10. Expression bodied constructors and finalizers
11. Expression bodied getters and setters
12. Throw can also be used as expression

1. Async main
2. Default literal expressions
3. Inferred tuple element names

1. Reference semantics with value types
2. Non-trailing named arguments
3. Leading underscores in numeric literals
4. private protected access modifier

1.4. Syntax

The core syntax of C# language is similar to that of other C-style languages such as C, C++ and Java. In particular:

1. Semicolons are used to denote the end of a statement.
2. Curly brackets are used to group statements. Statements are commonly grouped into methods (functions), methods into classes, and classes into namespaces.
3. Variables are assigned using an equals sign, but compared using two consecutive equals signs.
4. Square brackets are used with arrays, both to declare them and to get a value at a given index in one of them.

1.5. Distinguishing features

Some notable features of C# that distinguish it from C, C++, and Java where noted, are:

By design, C# is the programming language that most directly reflects the underlying Common

Language Infrastructure (CLI). Most of its intrinsic types correspond to value-types implemented by the CLI framework. However, the language specification does not state the code generation requirements of the compiler: that is, it does not state that a C# compiler must target a Common Language Runtime, or generate Common Intermediate Language (CIL), or generate any other specific format. Theoretically, a C# compiler could generate machine code like traditional compilers of C++ or Fortran.

C# supports strongly typed implicit variable declarations with the keyword var, and implicitly typed arrays with the keyword new followed by a collection initializer.

C# supports a strict Boolean data type, bool. Statements that take conditions, such as while and if, require an expression of a type that implements the true operator, such as the Boolean type. While C++ also has a Boolean type, it can be freely converted to and from integers, and expressions such as if(a) require only that a is convertible to bool, allowing a to be an int, or a pointer. C# disallows this "integer meaning true or false" approach, on the grounds that forcing programmers to use expressions that return exactly bool can prevent certain types of programming mistakes such as if (a b) (use of assignment instead of equality , which while not an error in C or C++, will be caught by the compiler anyway).

C# is more type safe than C++. The only implicit conversions by default are those that are considered safe, such as widening of integers. This is enforced at compile-time, during JIT, and, in some cases, at runtime. No implicit conversions occur between Booleans and integers, nor between enumeration members and integers (except for literal 0, which can be implicitly converted to any enumerated type). Any user-defined conversion must be explicitly marked as explicit or implicit, unlike C++ copy constructors and conversion operators, which are both implicit by default.

C# has explicit support for covariance and contravariance in generic types, unlike C++ which has some degree of support for contravariance simply through the semantics of return types on virtual methods.

Enumeration members are placed in their own scope.

The C# language does not allow for global variables or functions. All methods and members must be declared within classes. Static members of public classes can substitute for global variables and functions.

Local variables cannot shadow variables of the enclosing block, unlike C and C++.

Metaprogramming via C# attributes is part of the language. Many of these attributes duplicate the functionality of GCC's and VisualC++'s platform-dependent preprocessor directives.

Methods in programming language are the members of a class in a project, some methods have signatures and some don't have signatures. Methods can be void or can return something like string, integer, double, decimal, float and bool. If a method is void it means that the method does not return any data type.

Like C++, and unlike Java, C# programmers must use the keyword virtual to allow methods to be overridden by subclasses.

Extension methods in C# allow programmers to use static methods as if they were methods from a class's method table, allowing programmers to add methods to an object that they feel should exist on that object and its derivatives.

The type dynamic allows for run-time method binding, allowing for JavaScript-like method calls and run-time object composition.

C# has support for strongly-typed function pointers via the keyword delegate. Like the Qt framework's pseudo-C++ signal and slot, C# has semantics specifically surrounding publish-subscribe style events, though C# uses delegates to do so.

C# offers Java-like synchronized method calls, via the attribute , and has support for mutually-exclusive locks via the keyword lock.

C# provides properties as syntactic sugar for a common pattern in which a pair of methods, accessor (getter) and mutator (setter) encapsulate operations on a single attribute of a class. No redundant method signatures for the getter/setter implementations need be written, and the property may be accessed using attribute syntax rather than more verbose method calls.

A C# namespace provides the same level of code isolation as a Java package or a C++ , with very similar rules and features to a package.

In C#, memory address pointers can only be used within blocks specifically marked as unsafe, and programs with unsafe code need appropriate permissions to run. Most object access is done through safe object references, which always either point to a "live" object or have the well-defined null value; it is impossible to obtain a reference to a "dead" object (one that has been garbage collected), or to a random block of memory. An unsafe pointer can point to an instance of a value-type, array, string, or a block of memory allocated on a stack. Code that is not marked as unsafe can still store and manipulate pointers through the System.IntPtr type, but it cannot dereference them.

Managed memory cannot be explicitly freed; instead, it is automatically garbage collected.

Garbage collection addresses the problem of memory leaks by freeing the programmer of responsibility for releasing memory that is no longer needed.

Checked exceptions are not present in C# (in contrast to Java). This has been a conscious decision based on the issues of scalability and versionability.

Unlike C++, C# does not support multiple inheritance, although a class can implement any number of interfaces. This was a design decision by the language's lead architect to avoid complication and simplify architectural requirements throughout CLI. When implementing multiple interfaces that contain a method with the same signature, C# allows implementing each method depending on which interface that method is being called through, or, like Java, allows implementing the method once, and have that be the one invocation on a call through any of the class's interfaces.

However, unlike Java, C# supports operator overloading. Only the most commonly overloaded operators in C++ may be overloaded in C#.

C# has the ability to utilize LINQ through the Microsoft.NET Framework with the IEnumerable Interface a developer can query any .NET collection class, XML documents, ADO.NET datasets, and SQL databases. X. D. Zhang et al., "Research of the Database Access Technology Under.NET Framework", Applied Mechanics and Materials, Vols. 644-650, pp. 3077-3080, 2014 There are some advantages to using LINQ in C# and they are as follows: intellisense support, strong filtering capabilities, type safety with compile error checking ability, and brings consistency for querying data over a variety of sources.Otey, M. (2006, 02). LINQ to the future. SQL Server Magazine, 8, 17-21. Retrieved from http://ezaccess.libraries.psu.edu/login?url https://search-proquest-com.ezaccess.libraries.psu.edu/docview/214859896?accountid 13158 There are several different language structures that can be utilized with C# with LINQ and they are query expressions, lambda expressions, anonymous types, implicitly typed variables, extension methods, and object initializers. Sheldon, W. (2010, 11). New features in LINQ. SQL Server Magazine, 12, 37-40. Retrieved from http://ezaccess.libraries.psu.edu/login?url https://search-proquest-com.ezaccess.libraries.psu.edu/docview/770609095?accountid 13158

Though primarily an imperative language, C# 2.0 offered limited support for functional programming through first-class functions and closures in the form of anonymous delegates. C# 3.0 expanded support for functional programming with the introduction of a lightweight syntax for lambda expressions, extension methods (an affordance for modules), and a list comprehension syntax in the form of a "query comprehension" language.

1.6. Common type system

C# has a unified type system. This unified type system is called Common Type System (CTS).

A unified type system implies that all types, including primitives such as integers, are subclasses of the class. For example, every type inherits a method.

CTS separates data types into two categories:

Instances of value types do not have referential identity nor referential comparison semantics - equality and inequality comparisons for value types compare the actual data values within the instances, unless the corresponding operators are overloaded. Value types are derived from , always have a default value, and can always be created and copied. Some other limitations on value types are that they cannot derive from each other (but can implement interfaces) and cannot have an explicit default (parameterless) constructor. Examples of value types are all primitive types, such as (a signed 32-bit integer), (a 32-bit IEEE floating-point number), (a 16-bit Unicode code unit), and (identifies a specific point in time with nanosecond precision). Other examples are (enumerations) and (user defined structures).

In contrast, reference types have the notion of referential identity - each instance of a reference type is inherently distinct from every other instance, even if the data within both instances is the same. This is reflected in default equality and inequality comparisons for reference types, which test for referential rather than structural equality, unless the corresponding operators are overloaded (such as the case for). In general, it is not always possible to create an instance of a reference type, nor to copy an existing instance, or perform a value comparison on two existing instances, though specific reference types can provide such services by exposing a public constructor or implementing a corresponding interface (such as or). Examples of reference types are (the ultimate base class for all other C# classes), (a string of Unicode characters), and (a base class for all C# arrays).

Both type categories are extensible with user-defined types.

Boxing is the operation of converting a value-type object into a value of a corresponding reference type. Boxing in C# is implicit.

Unboxing is the operation of converting a value of a reference type (previously boxed) into a value of a value type. Unboxing in C# requires an explicit type cast. A boxed object of type T can only be unboxed to a T (or a nullable T).

Example:

```
int foo   42;         // Value type.
```

```
object bar   foo;     // foo is boxed to bar.
int foo2   (int)bar;  // Unboxed back to value type.
```

1. Reference types
2. Value types

1.7. Libraries

The C# specification details a minimum set of types and class libraries that the compiler expects to have available. In practice, C# is most often used with some implementation of the Common Language Infrastructure (CLI), which is standardized as ECMA-335 Common Language Infrastructure (CLI).

1.8. Examples

The following is a very simple C# program, a version of the classic "Hello world" example:

```
using System;

class Program
{

static void Main(string args)
  {
     Console.WriteLine("Hello, world!");
  }

}
```

What will display on the program is:

```
Hello, world!
```

Each line has a purpose:

using System;

The above line of code tells the compiler to use System as a candidate prefix for types used in the source code. In this case, when the compiler sees use of the Console type later in the source code, it tries to find a type named Console, first in the current assembly, followed by all referenced assemblies. In this case the compiler fails to find such a type, since the name of the type is actually System.Console. The compiler then attempts to find a type named System.Console by using the System prefix from the statement, and this time it succeeds. The statement allows the programmer to state all candidate prefixes to use during compilation instead of always using full type names.

class Program

Above is a class definition. Everything between the following pair of braces describes .

static void Main(string args)

This declares the class member method where the program begins execution. The .NET runtime calls the method. (Note: may also be called from elsewhere, like any other method, e.g. from another method of .) The static keyword makes the method accessible without an instance of . Each console application's entry point must be declared . Otherwise, the program would require an instance, but any instance would require a program. To avoid that irresolvable circular dependency, C# compilers processing console applications (like that above) report an error, if there is no method. The keyword declares that has no return value.

Console.WriteLine("Hello, world!");

This line writes the output. is a static class in the namespace. It provides an interface to the standard input, output, and error streams for console applications. The program calls the method , which displays on the console a line with the argument, the string .

A GUI example:

using System.Windows.Forms;

class Program
{

static void Main(string args)
  {
    MessageBox.Show("Hello, World!");

```
        System.Console.WriteLine("Is almost the same argument!");
    }

}
```

This example is similar to the previous example, except that it generates a dialog box that contains the message "Hello, World!" instead of writing it to the console.

1.9. Standardization and licensing

In August 2001, Microsoft Corporation, Hewlett-Packard and Intel Corporation co-sponsored the submission of specifications for C# as well as the Common Language Infrastructure (CLI) to the standards organization Ecma International.
In December 2001, ECMA released ECMA-334 C# Language Specification. C# became an ISO standard in 2003 (ISO/IEC 23270:2003 - Information technology — Programming languages — C#). ECMA had previously adopted equivalent specifications as the 2nd edition of C#, in December 2002.

In June 2005, ECMA approved edition 3 of the C# specification, and updated ECMA-334. Additions included partial classes, anonymous methods, nullable types, and generics (somewhat similar to C++ templates).

In July 2005, ECMA submitted to ISO/IEC JTC 1, via the latter's Fast-Track process, the standards and related TRs. This process usually takes 6–9 months.

The C# language definition and the CLI are standardized under ISO and Ecma standards that provide reasonable and non-discriminatory licensing protection from patent claims.

Microsoft has agreed not to sue open source developers for violating patents in non-profit projects for the part of the framework that is covered by the OSP. Microsoft has also agreed not to enforce patents relating to Novell products against Novell's paying customers with the exception of a list of products that do not explicitly mention C#, .NET or Novell's implementation of .NET (The Mono Project). However, Novell maintains that Mono does not infringe any Microsoft patents. Microsoft has also made a specific agreement not to enforce patent rights related to the Moonlight browser plugin, which depends on Mono, provided it is obtained through Novell.

1.10. Implementations

Microsoft is leading the development of the open-source reference C# compiler and set of tools, previously codenamed "Roslyn". The compiler, which is entirely written in managed code (C#), has been opened up and functionality surfaced as APIs. It is thus enabling developers to create refactoring and diagnostics tools.https://github.com/dotnet/roslynhttps://docs.microsoft.com/en-us/dotnet/articles/csharp/csharp While other implementations of C# exist, Visual C# is by far the one most commonly used.Watson, K et al., (2010). Beginning Visual C# 2010. Indianapolis, Indiana: Wiley. The Unity game engine uses C# as its primary scripting language.

Other C# compilers, which often including an implementation of the Common Language Infrastructure and the .NET class libraries up to .NET 2.0:

1. The Mono project provides an open-source C# compiler, a complete open-source implementation of the Common Language Infrastructure including the required framework libraries as they appear in the ECMA specification, and a nearly complete implementation of the Microsoft proprietary .NET class libraries up to .NET 3.5. As of Mono 2.6, no plans exist to implement WPF; WF is planned for a later release; and there are only partial implementations of LINQ to SQL and WCF.
2. The DotGNU project (now discontinued) also provided an open-source C# compiler, a nearly complete implementation of the Common Language Infrastructure including the required framework libraries as they appear in the ECMA specification, and subset of some of the remaining Microsoft proprietary .NET class libraries up to .NET 2.0 (those not documented or included in the ECMA specification, but included in Microsoft's standard .NET Framework distribution).
3. Microsoft's Shared Source Common Language Infrastructure, codenamed "Rotor", provides a shared source implementation of the CLR runtime and a C# compiler licensed for educational and research use only, and a subset of the required Common Language Infrastructure framework libraries in the ECMA specification (up to C# 2.0, and supported on Windows XP only).

2. syntax error

2.1.

Syntax error in a scientific calculator
In computer science, a syntax error is an error in the syntax of a sequence of characters or tokens that is intended to be written in a particular programming language.

For compiled languages, syntax errors are detected at compile-time. A program will not compile until all syntax errors are corrected. For interpreted languages, however, a syntax error may be detected during program execution, and an interpreter's error messages might not differentiate syntax errors from errors of other kinds.

There is some disagreement as to just what errors are "syntax errors". For example, some would say that the use of an uninitialized variable's value in Java code is a syntax error, but many others would disagreeIssue of syntax or semantics? and would classify this as a (static) semantic error.

In 8-bit home computers that used BASIC interpreter as their primary user interface, the SYNTAX ERROR error message became somewhat notorious, as this was the response to any command or user input the interpreter could not parse.

A syntax error may also occur when an invalid equation is entered into a calculator. This can be caused, for instance, by opening brackets without closing them, or less commonly, entering several decimal points in one number.

In Java the following is a syntactically correct statement:

System.out.println("Hello World");

while the following is not:

System.out.println(Hello World);

The second example would theoretically print the variable Hello World instead of the words Hello World. However, a variable in Java cannot have a space in between, so the syntactically correct line would be System.out.println(Hello_World).

A compiler will flag a syntax error when given source code that does not meet the requirements of the language grammar.

Type errors (such as an attempt to apply the ++ increment operator to a boolean variable in Java)

and undeclared variable errors are sometimes considered to be syntax errors when they are detected at compile-time. However, it is common to classify such errors as (static) semantic errors instead.Semantic Errors in Java Section 4.1.3: Syntax Error Handling, pp.194–195. Exercise 1.3, pp.27–28.

3. Syntax (programming languages)

3.1.

Syntax highlighting and indent style are often used to aid programmers in recognizing elements of source code. Color coded highlighting is used in this piece of code written in Python.

In computer science, the syntax of a computer language is the set of rules that defines the combinations of symbols that are considered to be a correctly structured document or fragment in that language. This applies both to programming languages, where the document represents source code, and markup languages, where the document represents data. The syntax of a language defines its surface form. Text-based computer languages are based on sequences of characters, while visual programming languages are based on the spatial layout and connections between symbols (which may be textual or graphical). Documents that are syntactically invalid are said to have a syntax error.

Syntax – the form – is contrasted with semantics – the meaning. In processing computer languages, semantic processing generally comes after syntactic processing, but in some cases semantic processing is necessary for complete syntactic analysis, and these are done together or concurrently. In a compiler, the syntactic analysis comprises the frontend, while semantic analysis comprises the backend (and middle end, if this phase is distinguished).

3.2. Levels of syntax

Computer language syntax is generally distinguished into three levels:

Distinguishing in this way yields modularity, allowing each level to be described and processed separately, and often independently. First a lexer turns the linear sequence of characters into a linear sequence of tokens; this is known as "lexical analysis" or "lexing". Second the parser turns the linear sequence of tokens into a hierarchical syntax tree; this is known as "parsing" narrowly speaking. Thirdly the contextual analysis resolves names and checks types. This modularity is sometimes possible, but in many real-world languages an earlier step depends on a later step – for example, the lexer hack in C is because tokenization depends on context. Even in these cases, syntactical analysis is often seen as approximating this ideal model.

The parsing stage itself can be divided into two parts: the parse tree or "concrete syntax tree" which is determined by the grammar, but is generally far too detailed for practical use, and the abstract syntax tree (AST), which simplifies this into a usable form. The AST and contextual

analysis steps can be considered a form of semantic analysis, as they are adding meaning and interpretation to the syntax, or alternatively as informal, manual implementations of syntactical rules that would be difficult or awkward to describe or implement formally.

The levels generally correspond to levels in the Chomsky hierarchy. Words are in a regular language, specified in the lexical grammar, which is a Type-3 grammar, generally given as regular expressions. Phrases are in a context-free language (CFL), generally a deterministic context-free language (DCFL), specified in a phrase structure grammar, which is a Type-2 grammar, generally given as production rules in Backus–Naur form (BNF). Phrase grammars are often specified in much more constrained grammars than full context-free grammars, in order to make them easier to parse; while the LR parser can parse any DCFL in linear time, the simple LALR parser and even simpler LL parser are more efficient, but can only parse grammars whose production rules are constrained. In principle, contextual structure can be described by a context-sensitive grammar, and automatically analyzed by means such as attribute grammars, though in general this step is done manually, via name resolution rules and type checking, and implemented via a symbol table which stores names and types for each scope.

Tools have been written that automatically generate a lexer from a lexical specification written in regular expressions and a parser from the phrase grammar written in BNF: this allows one to use declarative programming, rather than need to have procedural or functional programming. A notable example is the lex-yacc pair. These automatically produce a concrete syntax tree; the parser writer must then manually write code describing how this is converted to an abstract syntax tree. Contextual analysis is also generally implemented manually. Despite the existence of these automatic tools, parsing is often implemented manually, for various reasons – perhaps the phrase structure is not context-free, or an alternative implementation improves performance or error-reporting, or allows the grammar to be changed more easily. Parsers are often written in functional languages, such as Haskell, or in scripting languages, such as Python or Perl, or in C or C++.

As an example, (add 1 1) is a syntactically valid Lisp program (assuming the 'add' function exists, else name resolution fails), adding 1 and 1. However, the following are invalid:

```
(_ 1 1)    lexical error: '_' is not valid
(add 1 1   parsing error: missing closing ')'
```

Note that the lexer is unable to identify the first error – all it knows is that, after producing the token LEFT_PAREN, '(' the remainder of the program is invalid, since no word rule begins with '_'. The second error is detected at the parsing stage: The parser has identified the "list" production rule due to the '(' token (as the only match), and thus can give an error message; in general it may be ambiguous.

Type errors and undeclared variable errors are sometimes considered to be syntax errors when

they are detected at compile-time (which is usually the case when compiling strongly-typed languages), though it is common to classify these kinds of error as semantic errors instead.Section 4.1.3: Syntax Error Handling, pp.194–195. Exercise 1.3, pp.27–28.Semantic Errors in Java

As an example, the Python code

'a' + 1

contains a type error because it adds a string literal to an integer literal. Type errors of this kind can be detected at compile-time: They can be detected during parsing (phrase analysis) if the compiler uses separate rules that allow "integerLiteral + integerLiteral" but not "stringLiteral + integerLiteral", though it is more likely that the compiler will use a parsing rule that allows all expressions of the form "LiteralOrIdentifier + LiteralOrIdentifier" and then the error will be detected during contextual analysis (when type checking occurs). In some cases this validation is not done by the compiler, and these errors are only detected at runtime.

In a dynamically typed language, where type can only be determined at runtime, many type errors can only be detected at runtime. For example, the Python code

a + b

is syntactically valid at the phrase level, but the correctness of the types of a and b can only be determined at runtime, as variables do not have types in Python, only values do. Whereas there is disagreement about whether a type error detected by the compiler should be called a syntax error (rather than a static semantic error), type errors which can only be detected at program execution time are always regarded as semantic rather than syntax errors.

1. Words – the lexical level, determining how characters form tokens;
2. Phrases – the grammar level, narrowly speaking, determining how tokens form phrases;
3. Context – determining what objects or variables names refer to, if types are valid, etc.

3.3. Syntax definition

Parse tree of Python code with inset tokenization

The syntax of textual programming languages is usually defined using a combination of regular expressions (for lexical structure) and Backus–Naur form (for grammatical structure) to inductively

specify syntactic categories (nonterminals) and terminal symbols. Syntactic categories are defined by rules called productions, which specify the values that belong to a particular syntactic category. Terminal symbols are the concrete characters or strings of characters (for example keywords such as define, if, let, or void) from which syntactically valid programs are constructed.

A language can have different equivalent grammars, such as equivalent regular expressions (at the lexical levels), or different phrase rules which generate the same language. Using a broader category of grammars, such as LR grammars, can allow shorter or simpler grammars compared with more restricted categories, such as LL grammar, which may require longer grammars with more rules. Different but equivalent phrase grammars yield different parse trees, though the underlying language (set of valid documents) is the same.

Below is a simple grammar, defined using the notation of regular expressions and Extended Backus–Naur form. It describes the syntax of S-expressions, a data syntax of the programming language Lisp, which defines productions for the syntactic categories expression, atom, number, symbol, and list:

```
expression   atom   | list
atom         number | symbol
number       ?+
symbol       .*
list         '(', expression*, ')'
```

This grammar specifies the following:

Here the decimal digits, upper- and lower-case characters, and parentheses are terminal symbols.

The following are examples of well-formed token sequences in this grammar: '12345', '()', '(a b c232 (1))'

The grammar needed to specify a programming language can be classified by its position in the Chomsky hierarchy. The phrase grammar of most programming languages can be specified using a Type-2 grammar, i.e., they are context-free grammars, Section 2.2: Pushdown Automata, pp.101–114. though the overall syntax is context-sensitive (due to variable declarations and nested scopes), hence Type-1. However, there are exceptions, and for some languages the phrase grammar is Type-0 (Turing-complete).

In some languages like Perl and Lisp the specification (or implementation) of the language allows constructs that execute during the parsing phase. Furthermore, these languages have constructs that allow the programmer to alter the behavior of the parser. This combination effectively blurs the

distinction between parsing and execution, and makes syntax analysis an undecidable problem in these languages, meaning that the parsing phase may not finish. For example, in Perl it is possible to execute code during parsing using a BEGIN statement, and Perl function prototypes may alter the syntactic interpretation, and possibly even the syntactic validity of the remaining code.The following discussions give examples:

Colloquially this is referred to as "only Perl can parse Perl" (because code must be executed during parsing, and can modify the grammar), or more strongly "even Perl cannot parse Perl" (because it is undecidable). Similarly, Lisp macros introduced by the defmacro syntax also execute during parsing, meaning that a Lisp compiler must have an entire Lisp run-time system present. In contrast, C macros are merely string replacements, and do not require code execution.TEMPLATE[cite_web, url http://www.apl.jhu.edu/~hall/Lisp-Notes/Macros.html, title An Introduction to Common Lisp Macros, publisher Apl.jhu.edu, date 1996-02-08, accessdate 2013-08-17]

1. an expression is either an atom or a list;
2. an atom is either a number or a symbol;
3. a number is an unbroken sequence of one or more decimal digits, optionally preceded by a plus or minus sign;
4. a symbol is a letter followed by zero or more of any characters (excluding whitespace); and
5. a list is a matched pair of parentheses, with zero or more expressions inside it.

1. Perl and Undecidability
2. LtU comment clarifying that the undecidable problem is membership in the class of Perl programs
3. chromatic's example of Perl code that gives a syntax error depending on the value of random variable

3.4. Syntax versus semantics

The syntax of a language describes the form of a valid program, but does not provide any information about the meaning of the program or the results of executing that program. The meaning given to a combination of symbols is handled by semantics (either formal or hard-coded in a reference implementation). Not all syntactically correct programs are semantically correct. Many syntactically correct programs are nonetheless ill-formed, per the language's rules; and may (depending on the language specification and the soundness of the implementation) result in an error on translation or execution. In some cases, such programs may exhibit undefined behavior.

Even when a program is well-defined within a language, it may still have a meaning that is not intended by the person who wrote it.

Using natural language as an example, it may not be possible to assign a meaning to a grammatically correct sentence or the sentence may be false:

The following C language fragment is syntactically correct, but performs an operation that is not semantically defined (because p is a null pointer, the operations p->real and p->im have no meaning):

```
complex *p   NULL;
complex abs_p   sqrt (p->real * p->real + p->im * p->im);
```

As a simpler example,

```
int x;
printf("%d", x);
```

is syntactically valid, but not semantically defined, as it uses an uninitialized variable. Even though compilers for some programming languages (e.g., Java and C#) would detect uninitialized variable errors of this kind, they should be regarded as semantic errors rather than syntax errors.Issue of syntax or semantics?

1. "Colorless green ideas sleep furiously." is grammatically well formed but has no generally accepted meaning.
2. "John is a married bachelor." is grammatically well formed but expresses a meaning that cannot be true.

4. Java (programming language)

4.1.

TEMPLATE[Infobox_programming_language, name Java, logo , paradigm : (), , , , , , year (TEMPLATE)<ref>(TEMPLATE)</ref>, designer , developer (now owned by), typing , , , implementations Compilers: (javac, sjavac), (GCJ), Eclipse Compiler for Java (ECJ)

Virtual Machines: , , Azul Zing, , , Gluon VM, ,
JIT-Compilers: Oracle Graal, Azul Falcon (LLVM), influenced_by , ,<ref>(TEMPLATE)</ref> ,<ref>Java 5.0 added several new language features (the , , and), after they were introduced in the similar (and competing) language. </ref> ,<ref>(TEMPLATE)</ref> , ,<ref>(TEMPLATE)</ref> ,<ref>(TEMPLATE)</ref> ,<ref> stated on a number of public occasions, e.g. in a lecture at the Polytechnic Museum, Moscow in September 2005 (several independent first-hand accounts in Russian exist, e.g. one with an audio recording: (TEMPLATE)), that the Sun Java design team licensed the Oberon compiler sources a number of years prior to the release of Java and examined it: a (relative) compactness, type safety, garbage collection, no multiple inheritance for classes(TEMPLATE) all these key overall design features are shared by Java and Oberon.</ref> ,<ref> cites as a strong influence on the design of the Java programming language, stating that notable direct derivatives include Java interfaces (derived from Objective-C's) and primitive wrapper classes. [http://cs.gmu.edu/~sean/stuff/java-objc.html]</ref> ,<ref>(TEMPLATE)</ref><ref>(TEMPLATE)</ref> <ref>In the summer of 1996, Sun was designing the precursor to what is now the event model of the AWT and the JavaBeans TM component architecture. Borland contributed greatly to this process. We looked very carefully at Delphi Object Pascal and built a working prototype of bound method references in order to understand their interaction with the Java programming language and its APIs.</ref>, influenced , , , ,<ref name "chplspec">(TEMPLATE)</ref> , , ,<ref name "gambas">(TEMPLATE)</ref> , ,<ref>(TEMPLATE)</ref> , , , , , , , , , dialects , , license , , website (TEMPLATE), file_ext .java, , , wikibooks Java Programming]

Java is a general-purpose computer-programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.TEMPLATE[cite_web, url http://www.oracle.com/technetwork/java/intro-141325.html, title 1.2 Design Goals of the Java™ Programming Language, publisher Oracle, date January 1, 1999, accessdate 2013-01-14] Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture. As of 2016, Java is one of the most popular programming languages in use,TEMPLATE[cite_web, url https://www.wired.com/2013/01/java-no-longer-a-favorite/, title Is Java Losing Its Mojo?, quote Java is on the wane, at least according to one outfit that keeps on eye on the ever-changing world

of computer programming languages. For more than a decade, it has dominated the Programming Community Index, and is back on top – a snapshot of software developer enthusiasm that looks at things like internet search results to measure how much buzz different languages have. But lately, Java has been slipping., first Robert, last McMillan, date August 1, 2013, publisher]RedMonk Index on redmonk.com (Stephen O'Grady, January 2015) particularly for client-server web applications, with a reported 9 million developers. Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

The original and reference implementation Java compilers, virtual machines, and class libraries were originally released by Sun under proprietary licenses. As of May 2007, in compliance with the specifications of the Java Community Process, Sun relicensed most of its Java technologies under the GNU General Public License. Others have also developed alternative implementations of these Sun technologies, such as the GNU Compiler for Java (bytecode compiler), GNU Classpath (standard libraries), and IcedTea-Web (browser plugin for applets).

The latest version is Java 9, released on September 21, 2017, and is one of the two versions currently supported for free by Oracle. Versions earlier than Java 8 are supported by companies on a commercial basis; e.g. by Oracle back to Java 6 as of October 2017 (while they still "highly recommend that you uninstall"https://www.java.com/en/download/faq/remove_olderversions.xml pre-Java 8 from at least Windows computers).

4.2. History

Duke, the Java mascot
James Gosling, the creator of Java (2008)
The TIOBE programming language popularity index graph from 2002 to 2015. Over the course of a decade Java (blue) and C (black) competing for the top position.

James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. Java was originally designed for interactive television, but it was too advanced for the digital cable television industry at the time.Object-oriented programming The language was initially called Oak after an oak tree that stood outside Gosling's office. Later the project went by the name Green and was finally renamed Java, from Java coffee."So why did they decide to call it Java?", Kieron Murphy, JavaWorld.com, 10/04/96 Gosling designed Java with a C/C++-style syntax that system and application programmers would find familiar.Kabutz, Heinz; Once Upon an Oak. Artima. Retrieved April 29, 2007.

Sun Microsystems released the first public implementation as Java 1.0 in 1995. It promised "Write Once, Run Anywhere" (WORA), providing no-cost run-times on popular platforms. Fairly secure and featuring configurable security, it allowed network- and file-access restrictions. Major web browsers soon incorporated the ability to run Java applets within web pages, and Java quickly became popular. The Java 1.0 compiler was re-written in Java by Arthur van Hoff to comply strictly with the Java 1.0 language specification. With the advent of Java 2 (released initially as J2SE 1.2 in December 1998 – 1999), new versions had multiple configurations built for different types of platforms. J2EE included technologies and APIs for enterprise applications typically run in server environments, while J2ME featured APIs optimized for mobile applications. The desktop version was renamed J2SE. In 2006, for marketing purposes, Sun renamed new J2 versions as Java EE, Java ME, and Java SE, respectively.

In 1997, Sun Microsystems approached the ISO/IEC JTC 1 standards body and later the Ecma International to formalize Java, but it soon withdrew from the process.TEMPLATE[cite_web, url http://www.open-std.org/JTC1/SC22/JSG/, title JSG – Java Study Group, work open-std.org]TEMPLATE[cite_web, url http://csdl2.computer.org/comp/proceedings/hicss/2001/0981/05/09815015.pdf, title Why Java™ Was – Not – Standardized Twice]TEMPLATE[cite_web, url http://www.zdnet.com/news/what-is-ecma-and-why-microsoft-cares/298821, title What is ECMA—and why Microsoft cares] Java remains a de facto standard, controlled through the Java Community Process. At one time, Sun made most of its Java implementations available without charge, despite their proprietary software status. Sun generated revenue from Java through the selling of licenses for specialized products such as the Java Enterprise System.

On November 13, 2006, Sun released much of its Java virtual machine (JVM) as free and open-source software, (FOSS), under the terms of the GNU General Public License (GPL). On May 8, 2007, Sun finished the process, making all of its JVM's core code available under free software/open-source distribution terms, aside from a small portion of code to which Sun did not hold the copyright.TEMPLATE[cite_web, url http://grnlight.net/index.php/programming-articles/115-javaone-sun-the-bulk-of-java-is-open-sourced, title JAVAONE: Sun – The bulk of Java is open sourced, publisher GrnLight.net, accessdate 2014-05-26]

Sun's vice-president Rich Green said that Sun's ideal role with regard to Java was as an "evangelist". Following Oracle Corporation's acquisition of Sun Microsystems in 2009–10, Oracle has described itself as the "steward of Java technology with a relentless commitment to fostering a community of participation and transparency". This did not prevent Oracle from filing a lawsuit against Google shortly after that for using Java inside the Android SDK (see Google section below). Java software runs on everything from laptops to data centers, game consoles to scientific supercomputers. On April 2, 2010, James Gosling resigned from Oracle.

In January 2016, Oracle announced that Java runtime environments based on JDK 9 will discontinue the browser plugin.

There were five primary goals in the creation of the Java language:

, both Java 8 and 9 are officially supported. Major release versions of Java, along with their release dates:

1. It must be "simple, object-oriented, and familiar".
2. It must be "robust and secure".
3. It must be "architecture-neutral and portable".
4. It must execute with "high performance".
5. It must be "interpreted, threaded, and dynamic".

1. JDK 1.0 (January 23, 1996)
2. JDK 1.1 (February 19, 1997)
3. J2SE 1.2 (December 8, 1998)
4. J2SE 1.3 (May 8, 2000)
5. J2SE 1.4 (February 6, 2002)
6. J2SE 5.0 (September 30, 2004)
7. Java SE 6 (December 11, 2006)
8. Java SE 7 (July 28, 2011)
9. Java SE 8 (March 18, 2014)
10. Java SE 9 (September 21, 2017)

4.3. Practices

One design goal of Java is portability, which means that programs written for the Java platform must run similarly on any combination of hardware and operating system with adequate runtime support.
This is achieved by compiling the Java language code to an intermediate representation called Java bytecode, instead of directly to architecture-specific machine code. Java bytecode instructions are analogous to machine code, but they are intended to be executed by a virtual machine (VM) written specifically for the host hardware. End users commonly use a Java Runtime Environment (JRE) installed on their own machine for standalone Java applications, or in a web browser for Java applets.

Standard libraries provide a generic way to access host-specific features such as graphics, threading, and networking.

The use of universal bytecode makes porting simple. However, the overhead of interpreting bytecode into machine instructions made interpreted programs almost always run more slowly than native executables. Just-in-time (JIT) compilers that compile bytecodes to machine code during runtime were introduced from an early stage. Java itself is platform-independent and is adapted to the particular platform it is to run on by a Java virtual machine for it, which translates the Java bytecode into the platform's machine language.

Oracle Corporation is the current owner of the official implementation of the Java SE platform, following their acquisition of Sun Microsystems on January 27, 2010. This implementation is based on the original implementation of Java by Sun. The Oracle implementation is available for Microsoft Windows (still works for XP, while only later versions are currently officially supported), macOS, Linux, and Solaris. Because Java lacks any formal standardization recognized by Ecma International, ISO/IEC, ANSI, or other third-party standards organization, the Oracle implementation is the de facto standard.

The Oracle implementation is packaged into two different distributions: The Java Runtime Environment (JRE) which contains the parts of the Java SE platform required to run Java programs and is intended for end users, and the Java Development Kit (JDK), which is intended for software developers and includes development tools such as the Java compiler, Javadoc, Jar, and a debugger.

OpenJDK is another notable Java SE implementation that is licensed under the GNU GPL. The implementation started when Sun began releasing the Java source code under the GPL. As of Java SE 7, OpenJDK is the official Java reference implementation.

The goal of Java is to make all implementations of Java compatible. Historically, Sun's trademark license for usage of the Java brand insists that all implementations be "compatible". This resulted in a legal dispute with Microsoft after Sun claimed that the Microsoft implementation did not support RMI or JNI and had added platform-specific features of their own. Sun sued in 1997, and, in 2001, won a settlement of US$20 million, as well as a court order enforcing the terms of the license from Sun. As a result, Microsoft no longer ships Java with Windows.

Platform-independent Java is essential to Java EE, and an even more rigorous validation is required to certify an implementation. This environment enables portable server-side applications.

Programs written in Java have a reputation for being slower and requiring more memory than those written in C++. However, Java programs' execution speed improved significantly with the introduction of just-in-time compilation in 1997/1998 for Java 1.1, the addition of language features

supporting better code analysis (such as inner classes, the StringBuilder class, optional assertions, etc.), and optimizations in the Java virtual machine, such as HotSpot becoming the default for Sun's JVM in 2000. With Java 1.5, the performance was improved with the addition of the java.util.concurrent package, including lock free implementations of the ConcurrentMaps and other multi-core collections, and it was improved further with Java 1.6.

Some platforms offer direct hardware support for Java; there are microcontrollers that can run Java in hardware instead of a software Java virtual machineTEMPLATE[Cite_journal, last Salcic, first Zoran, last2 Park, first2 Heejong, last3 Teich, first3 Jürgen, last4 Malik, first4 Avinash, last5 Nadeem, first5 Muhammad, date 2017-07-22, title Noc-HMP: A Heterogeneous Multicore Processor for Embedded Systems Designed in SystemJ, url http://dl.acm.org/citation.cfm?id 3097980.3073416, journal ACM Transactions on Design Automation of Electronic Systems (TODAES), volume 22, issue 4, pages 73, doi 10.1145/3073416, issn 1084-4309], and some ARM based processors could have hardware support for executing Java bytecode through their Jazelle option, though support has mostly been dropped in current implementations of ARM.

Java uses an automatic garbage collector to manage memory in the object lifecycle. The programmer determines when objects are created, and the Java runtime is responsible for recovering the memory once objects are no longer in use. Once no references to an object remain, the unreachable memory becomes eligible to be freed automatically by the garbage collector. Something similar to a memory leak may still occur if a programmer's code holds a reference to an object that is no longer needed, typically when objects that are no longer needed are stored in containers that are still in use. If methods for a nonexistent object are called, a "null pointer exception" is thrown.

One of the ideas behind Java's automatic memory management model is that programmers can be spared the burden of having to perform manual memory management. In some languages, memory for the creation of objects is implicitly allocated on the stack or explicitly allocated and deallocated from the heap. In the latter case, the responsibility of managing memory resides with the programmer. If the program does not deallocate an object, a memory leak occurs. If the program attempts to access or deallocate memory that has already been deallocated, the result is undefined and difficult to predict, and the program is likely to become unstable or crash. This can be partially remedied by the use of smart pointers, but these add overhead and complexity. Note that garbage collection does not prevent "logical" memory leaks, i.e., those where the memory is still referenced but never used.

Garbage collection may happen at any time. Ideally, it will occur when a program is idle. It is guaranteed to be triggered if there is insufficient free memory on the heap to allocate a new object; this can cause a program to stall momentarily. Explicit memory management is not possible in Java.

Java does not support C/C++ style pointer arithmetic, where object addresses and unsigned integers (usually long integers) can be used interchangeably. This allows the garbage collector to relocate referenced objects and ensures type safety and security.

As in C++ and some other object-oriented languages, variables of Java's primitive data types are either stored directly in fields (for objects) or on the stack (for methods) rather than on the heap, as is commonly true for non-primitive data types (but see escape analysis). This was a conscious decision by Java's designers for performance reasons.

Java contains multiple types of garbage collectors. By default, HotSpot uses the parallel scavenge garbage collector.TEMPLATE[cite_web, url http://docs.oracle.com/javase/7/docs/technotes/guides/vm/performance-enhancements-7.html, title Java HotSpot™ Virtual Machine Performance Enhancements, publisher Oracle.com, accessdate 2017-04-26] However, there are also several other garbage collectors that can be used to manage the heap. For 90% of applications in Java, the Concurrent Mark-Sweep (CMS) garbage collector is sufficient. Oracle aims to replace CMS with the Garbage-First collector (G1).

4.4. Syntax

upright 0.9|Dependency graph of the Java Core classes (created with jdeps and Gephi). The most frequently used classes Object and String appear in the centre of the diagram.

The syntax of Java is largely influenced by C++. Unlike C++, which combines the syntax for structured, generic, and object-oriented programming, Java was built almost exclusively as an object-oriented language. All code is written inside classes, and every data item is an object, with the exception of the primitive data types, (i.e. integers, floating-point numbers, boolean values, and characters), which are not objects for performance reasons. Java reuses some popular aspects of C++ (such as the method).

Unlike C++, Java does not support operator overloading or multiple inheritance for classes, though multiple inheritance is supported for interfaces.TEMPLATE[cite_web, url https://docs.oracle.com/javase/tutorial/java/IandI/multipleinheritance.html, title Multiple Inheritance of State, Implementation, and Type, publisher Oracle, work The Java™ Tutorials, accessdate December 10, 2014]

Java uses comments similar to those of C++. There are three different styles of comments: a single line style marked with two slashes , a multiple line style opened with /* and closed with */, and the Javadoc commenting style opened with /** and closed with */. The Javadoc style of commenting allows the user to run the Javadoc executable to create documentation for the

program and can be read by some integrated development environments (IDEs) such as Eclipse to allow developers to access documentation within the IDE.

Example:

```
// This is an example of a single line comment using two slashes

/* This is an example of a multiple line comment using the slash and asterisk.

This type of comment can be used to hold a lot of information or deactivate
code, but it is very important to remember to close the comment. */

package fibsandlies;
import java.util.HashMap;

/**

* This is an example of a Javadoc comment; Javadoc can compile documentation
* from this text. Javadoc comments must immediately precede the class, method, or field being documented.
*/

public class FibCalculator extends Fibonacci implements Calculator {

private static Map memoized   new HashMap();

  /*
   * The main method written as follows is used by the JVM as a starting point for the program.
   */
  public static void main(String args) {
      memoized.put(1, 1);
      memoized.put(2, 1);
      System.out.println(fibonacci(12)); //Get the 12th Fibonacci number and print to console
  }

  /**
   * An example of a method written in Java, wrapped in a class.
   * Given a non-negative number FIBINDEX, returns
   * the Nth Fibonacci number, where N equals FIBINDEX.
   * @param fibIndex The index of the Fibonacci number
   * @return The Fibonacci number
```

```
    */
   public static int fibonacci(int fibIndex) {
      if (memoized.containsKey(fibIndex)) {
         return memoized.get(fibIndex);
      } else {
         int answer   fibonacci(fibIndex - 1) + fibonacci(fibIndex - 2);
         memoized.put(fibIndex, answer);
         return answer;
      }
   }

}
```

4.5. "Hello world" example

The traditional "Hello, world!" program can be written in Java as:TEMPLATE[cite_web, url http://download.oracle.com/javase/tutorial/getStarted/application/index.html, title Lesson: A Closer Look at the "Hello World!" Application, work The Java™ Tutorials > Getting Started, publisher , accessdate 2011-04-14]

```
class HelloWorldApp {

public static void main(String args) {
      System.out.println("Hello World!"); // Prints the string to the console.
   }

}
```

Source files must be named after the public class they contain, appending the suffix .java, for example, HelloWorldApp.java. It must first be compiled into bytecode, using a Java compiler, producing a file named HelloWorldApp.class. Only then can it be executed, or "launched". The Java source file may only contain one public class, but it can contain multiple classes with other than public access modifier and any number of public inner classes. When the source file contains multiple classes, make one class "public" and name the source file with that public class name.

A class that is not declared public may be stored in any .java file. The compiler will generate a class file for each class defined in the source file. The name of the class file is the name of the class, with .class appended. For class file generation, anonymous classes are treated as if their name were the concatenation of the name of their enclosing class, a $, and an integer.

The keyword public denotes that a method can be called from code in other classes, or that a class may be used by classes outside the class hierarchy. The class hierarchy is related to the name of the directory in which the .java file is located. This is called an access level modifier. Other access level modifiers include the keywords private and protected.

The keyword static in front of a method indicates a static method, which is associated only with the class and not with any specific instance of that class. Only static methods can be invoked without a reference to an object. Static methods cannot access any class members that are not also static. Methods that are not designated static are instance methods and require a specific instance of a class to operate.

The keyword void indicates that the main method does not return any value to the caller. If a Java program is to exit with an error code, it must call System.exit() explicitly.

The method name "main" is not a keyword in the Java language. It is simply the name of the method the Java launcher calls to pass control to the program. Java classes that run in managed environments such as applets and Enterprise JavaBeans do not use or need a main() method. A Java program may contain multiple classes that have main methods, which means that the VM needs to be explicitly told which class to launch from.

The main method must accept an array of objects. By convention, it is referenced as args although any other legal identifier name can be used. Since Java 5, the main method can also use variable arguments, in the form of public static void main(String... args), allowing the main method to be invoked with an arbitrary number of String arguments. The effect of this alternate declaration is semantically identical (to the args parameter which is still an array of String objects), but it allows an alternative syntax for creating and passing the array.

The Java launcher launches Java by loading a given class (specified on the command line or as an attribute in a JAR) and starting its public static void main(String) method. Stand-alone programs must declare this method explicitly. The String args parameter is an array of objects containing any arguments passed to the class. The parameters to main are often passed by means of a command line.

Printing is part of a Java standard library: The class defines a public static field called . The out object is an instance of the class and provides many methods for printing data to standard out, including which also appends a new line to the passed string.

The string "Hello World!" is automatically converted to a String object by the compiler.

4.6. Special classes

Java applets are programs that are embedded in other applications, typically in a Web page displayed in a web browser.

```
// Hello.java
import javax.swing.JApplet;
import java.awt.Graphics;

public class Hello extends JApplet {

public void paintComponent(final Graphics g) {
      g.drawString("Hello, world!", 65, 95);
  }

}
```

The import statements direct the Java compiler to include the and classes in the compilation. The import statement allows these classes to be referenced in the source code using the simple class name (i.e. JApplet) instead of the fully qualified class name (FQCN, i.e. javax.swing.JApplet).

The Hello class extends (subclasses) the JApplet (Java Applet) class; the JApplet class provides the framework for the host application to display and control the lifecycle of the applet. The JApplet class is a JComponent (Java Graphical Component) which provides the applet with the capability to display a graphical user interface (GUI) and respond to user events.

The Hello class overrides the method (additionally indicated with the annotation, supported as of JDK 1.5, Override) inherited from the superclass to provide the code to display the applet. The paintComponent() method is passed a Graphics object that contains the graphic context used to display the applet. The paintComponent() method calls the graphic context method to display the "Hello, world!" string at a pixel offset of (65, 95) from the upper-left corner in the applet's display.

Hello World Applet

An applet is placed in an HTML document using the <applet> HTML element. The applet tag has three attributes set: code "Hello.class" specifies the name of the JApplet class and width "200" height "200" sets the pixel width and height of the applet. Applets may also be embedded in HTML using either the object or embed element, although support for these elements by web browsers is inconsistent. However, the applet tag is deprecated, so the object tag is preferred where supported.

The host application, typically a Web browser, instantiates the Hello applet and creates an for the applet. Once the applet has initialized itself, it is added to the AWT display hierarchy. The paintComponent() method is called by the AWT event dispatching thread whenever the display needs the applet to draw itself.

Java Servlet technology provides Web developers with a simple, consistent mechanism for extending the functionality of a Web server and for accessing existing business systems. Servlets are server-side Java EE components that generate responses (typically HTML pages) to requests (typically HTTP requests) from clients. A servlet can almost be thought of as an applet that runs on the server side—without a face.

```
// Hello.java
import java.io.*;
import javax.servlet.*;

public class Hello extends GenericServlet {

public void service(final ServletRequest request, final ServletResponse response)
  throws ServletException, IOException {
     response.setContentType("text/html");
     final PrintWriter pw   response.getWriter();
     try {
        pw.println("Hello, world!");
     } finally {
        pw.close();
     }
  }

}
```

The import statements direct the Java compiler to include all the public classes and interfaces from the and packages in the compilation. Packages make Java well suited for large scale applications.

The Hello class extends the class; the GenericServlet class provides the interface for the server to forward requests to the servlet and control the servlet's lifecycle.

The Hello class overrides the method defined by the interface to provide the code for the service request handler. The service() method is passed: a object that contains the request from the client and a object used to create the response returned to the client. The service() method declares

that it throws the exceptions and if a problem prevents it from responding to the request.

The method in the response object is called to set the MIME content type of the returned data to "text/html". The method in the response returns a object that is used to write the data that is sent to the client. The method is called to write the "Hello, world!" string to the response and then the method is called to close the print writer, which causes the data that has been written to the stream to be returned to the client.

JavaServer Pages (JSP) are server-side Java EE components that generate responses, typically HTML pages, to HTTP requests from clients. JSPs embed Java code in an HTML page by using the special delimiters and %>. A JSP is compiled to a Java servlet, a Java application in its own right, the first time it is accessed. After that, the generated servlet creates the response.

Swing is a graphical user interface library for the Java SE platform. It is possible to specify a different look and feel through the pluggable look and feel system of Swing. Clones of Windows, GTK+, and Motif are supplied by Sun. Apple also provides an Aqua look and feel for macOS. Where prior implementations of these looks and feels may have been considered lacking, Swing in Java SE 6 addresses this problem by using more native GUI widget drawing routines of the underlying platforms.

This example Swing application creates a single window with "Hello, world!" inside:

```
// Hello.java (Java SE 5)
import javax.swing.*;

public class Hello extends JFrame {

public Hello() {
      super("hello");
      super.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
      super.add(new JLabel("Hello, world!"));
      super.pack();
      super.setVisible(true);
  }

  public static void main(final String args) {
      new Hello();
  }

}
```

The first import includes all the public classes and interfaces from the package.

The Hello class extends the class; the JFrame class implements a window with a title bar and a close control.

The Hello() constructor initializes the frame by first calling the superclass constructor, passing the parameter "hello", which is used as the window's title. It then calls the method inherited from JFrame to set the default operation when the close control on the title bar is selected to this causes the JFrame to be disposed of when the frame is closed (as opposed to merely hidden), which allows the Java virtual machine to exit and the program to terminate. Next, a is created for the string "Hello, world!" and the method inherited from the superclass is called to add the label to the frame. The method inherited from the superclass is called to size the window and lay out its contents.

The main() method is called by the Java virtual machine when the program starts. It instantiates a new Hello frame and causes it to be displayed by calling the method inherited from the superclass with the boolean parameter true. Once the frame is displayed, exiting the main method does not cause the program to terminate because the AWT event dispatching thread remains active until all of the Swing top-level windows have been disposed.

In 2004, generics were added to the Java language, as part of J2SE 5.0. Prior to the introduction of generics, each variable declaration had to be of a specific type. For container classes, for example, this is a problem because there is no easy way to create a container that accepts only specific types of objects. Either the container operates on all subtypes of a class or interface, usually Object, or a different container class has to be created for each contained class. Generics allow compile-time type checking without having to create many container classes, each containing almost identical code. In addition to enabling more efficient code, certain runtime exceptions are prevented from occurring, by issuing compile-time errors. If Java prevented all runtime type errors (ClassCastException's) from occurring, it would be type safe.

In 2016, the type system was shown not to be safe at all, it was proven unsound.

4.7. Criticism

Criticisms directed at Java include the implementation of generics,. More comments to the original article available at earlier archive snapshots like . speed, the handling of unsigned numbers, the implementation of floating-point arithmetic,TEMPLATE[cite_web, last Kahan, first William, title How Java's Floating-Point Hurts Everyone Everywhere, url http://www.cs.berkeley.edu/~wkahan/JAVAhurt.pdf, publisher Electrical Engineering & Computer

Science, University of California at Berkeley, accessdate June 4, 2011] and a history of security vulnerabilities in the primary Java VM implementation HotSpot.

4.8. Use outside the Java platform

The Java programming language requires the presence of a software platform in order for compiled programs to be executed. Oracle supplies the Java platform for use with Java. The Android SDK is an alternative software platform, used primarily for developing Android applications.

The Android operating system makes extensive use of Java-related technology.

The Java language is a key pillar in Android, an open source mobile operating system. Although Android, built on the Linux kernel, is written largely in C, the Android SDK uses the Java language as the basis for Android applications. The bytecode language supported by the Android SDK is incompatible with Java bytecode and runs on its own virtual machine, optimized for low-memory devices such as smartphones and tablet computers.
Depending on the Android version, the bytecode is either interpreted by the Dalvik virtual machine or compiled into native code by the Android Runtime.

Android does not provide the full Java SE standard library, although the Android SDK does include an independent implementation of a large subset of it. It supports Java 6 and some Java 7 features, offering an implementation compatible with the standard library (Apache Harmony).

The use of Java-related technology in Android led to a legal dispute between Oracle and Google. On May 7, 2012, a San Francisco jury found that if APIs could be copyrighted, then Google had infringed Oracle's copyrights by the use of Java in Android devices. District Judge William Haskell Alsup ruled on May 31, 2012, that APIs cannot be copyrighted,

but this was reversed by the United States Court of Appeals for the Federal Circuit in May 2014. On May 26, 2016, the district court decided in favor of Google, ruling the copyright infringement of the Java API in Android constitutes fair use.TEMPLATE[cite_news, last1 Mullin, first1 Joe, title Google beats Oracle—Android makes "fair use" of Java APIs, url https://arstechnica.com/tech-policy/2016/05/google-wins-trial-against-oracle-as-jury-finds-android-is-fair-use/, accessdate 2016-05-26, publisher Ars Technica, date 2016-05-26]

4.9. Class libraries

The Java Class Library is the standard library, developed to support application development in Java. It is controlled by Sun Microsystems in cooperation with others through the Java Community Process program. Companies or individuals participating in this process can influence the design and development of the APIs. This process has been a subject of controversy. The class library contains features such as:

1. The core libraries, which include:
2. IO/NIO Networking Reflection Concurrency Generics Scripting/Compiler Functional programming (Lambda, Streaming) Collection libraries that implement data structures such as lists, dictionaries, trees, sets, queues and double-ended queue, or stacks XML Processing (Parsing, Transforming, Validating) libraries SecurityTEMPLATE[cite_web, url http://docs.oracle.com/javase/8/docs/technotes/guides/security/overview/jsoverview.html, title Java™ Security Overview, publisher Oracle, work Java Documentation, accessdate December 18, 2014] Internationalization and localization librariesTEMPLATE[cite_web, url http://docs.oracle.com/javase/tutorial/i18n/, title Trail: Internationalization, publisher Oracle, work The Java™ Tutorials, accessdate December 18, 2014]
3. The integration libraries, which allow the application writer to communicate with external systems. These libraries include:
4. The Java Database Connectivity (JDBC) API for database access Java Naming and Directory Interface (JNDI) for lookup and discovery RMI and CORBA for distributed application development JMX for managing and monitoring applications
5. User interface libraries, which include:
6. The (heavyweight, or native) Abstract Window Toolkit (AWT), which provides GUI components, the means for laying out those components and the means for handling events from those components The (lightweight) Swing libraries, which are built on AWT but provide (non-native) implementations of the AWT widgetry APIs for audio capture, processing, and playback JavaFX
7. A platform dependent implementation of the Java virtual machine that is the means by which the bytecodes of the Java libraries and third party applications are executed
8. Plugins, which enable applets to be run in web browsers
9. Java Web Start, which allows Java applications to be efficiently distributed to end users across the Internet
10. Licensing and documentation

4.10. Documentation

Javadoc is a comprehensive documentation system, created by Sun Microsystems, used by many Java developers. It provides developers with an organized system for documenting their code. Javadoc comments have an extra asterisk at the beginning, i.e. the delimiters are /** and */, whereas the normal multi-line comments in Java are set off with the delimiters /* and */.

4.11. Editions

Sun has defined and supports four editions of Java targeting different application environments and segmented many of its APIs so that they belong to one of the platforms. The platforms are:

The classes in the Java APIs are organized into separate groups called packages. Each package contains a set of related interfaces, classes, and exceptions. Refer to the separate platforms for a description of the packages available.

Sun also provided an edition called PersonalJava that has been superseded by later, standards-based Java ME configuration-profile pairings.

1. Java Card for smartcards.
2. Java Platform, Micro Edition (Java ME) – targeting environments with limited resources.
3. Java Platform, Standard Edition (Java SE) – targeting workstation environments.
4. Java Platform, Enterprise Edition (Java EE) – targeting large distributed enterprise or Internet environments.

5. javac

5.1.

javac (pronounced "java-see") is the primary Java compiler included in the Java Development Kit (JDK) from Oracle Corporation. Martin Odersky implemented the GJ compiler, and his implementation became the basis for javac.

The compiler accepts source code conforming to the Java language specification (JLS) and produces Java bytecode conforming to the Java Virtual Machine Specification (JVMS).

javac is itself written in Java. The compiler can also be invoked programmatically.
"an application can access javac programmatically."

5.2. History

On 13 November 2006, Sun's HotSpot Java virtual machine (JVM) and Java Development Kit (JDK) were made availableSun opens Java (feature story) under the GPL license.Sun's OpenJDK Hotspot page

Since version 0.95, GNU Classpath, a free implementation of the Java Class Library, supports compiling and running javac using the Classpath runtime — GNU Interpreter for Java (GIJ) — and compiler — GNU Compiler for Java (GCJ) — and also allows one to compile the GNU Classpath class library, tools and examples with javac itself.
"This release supports compiling and running the GPL OpenJDK javac compiler"

6. Generics in Java

6.1.

Generics are a facility of generic programming that were added to the Java programming language in 2004 within version J2SE 5.0. They were designed to extend Java's type system to allow “a type or method to operate on objects of various types while providing compile-time type safety”Java Programming Language. The aspect compile-time type safety was not fully achieved, since it was shown in 2016 that it is not guaranteed in all cases.A ClassCastException can be thrown even in the absence of casts or nulls.TEMPLATE[cite_web, url https://raw.githubusercontent.com/namin/unsound/master/doc/unsound-oopsla16.pdf, title Java and Scala’s Type Systems are Unsound]

The Java collections framework supports generics to specify the type of objects stored in a collection instance.

In 1998, Gilad Bracha, Martin Odersky, David Stoutamire and Philip Wadler created Generic Java, an extension to the Java language to support generic types.GJ: Generic Java Generic Java was incorporated in Java with the addition of wildcards.

6.2. Hierarchy and classification

According to Java Language Specification:Java Language Specification, Third Edition
by James Gosling, Bill Joy, Guy Steele, Gilad Bracha – Prentice Hall PTR 2005

1. A type variable is an unqualified identifier. Type variables are introduced by generic class declarations, generic interface declarations, generic method declarations, and by generic constructor declarations.
2. A class is generic if it declares one or more type variables. These type variables are known as the type parameters of the class. It defines one or more type variables that act as parameters. A generic class declaration defines a set of parameterized types, one for each possible invocation of the type parameter section. All of these parameterized types share the same class at runtime.
3. An interface is generic if it declares one or more type variables. These type variables are known as the type parameters of the interface. It defines one or more type variables that act as parameters. A generic interface declaration defines a set of types, one for each possible

invocation of the type parameter section. All parameterized types share the same interface at runtime.
4. A method is generic if it declares one or more type variables. These type variables are known as the formal type parameters of the method. The form of the formal type parameter list is identical to a type parameter list of a class or interface.
5. A constructor can be declared as generic, independently of whether the class that the constructor is declared in is itself generic. A constructor is generic if it declares one or more type variables. These type variables are known as the formal type parameters of the constructor. The form of the formal type parameter list is identical to a type parameter list of a generic class or interface.

6.3. Motivation

The following block of Java code illustrates a problem that exists when not using generics. First, it declares an ArrayList of type Object. Then, it adds a String to the ArrayList. Finally, it attempts to retrieve the added String and cast it to an Integer.

```
List v   new ArrayList();
v.add("test");
Integer i   (Integer)v.get(0); // Run time error
```

Although the code is compiled without error, it throws a runtime exception (java.lang.ClassCastException) when executing the third line of code. This type of problem can be avoided by using generics and is the primary motivation for using generics.

Using generics, the above code fragment can be rewritten as follows:

```
List v   new ArrayList();
v.add("test");
Integer i   v.get(0); // (type error)  compilation-time error
```

The type parameter String within the angle brackets declares the ArrayList to be constituted of String (a descendant of the ArrayList's generic Object constituents). With generics, it is no longer necessary to cast the third line to any particular type, because the result of v.get(0) is defined as String by the code generated by the compiler.

Compiling the third line of this fragment with J2SE 5.0 (or later) will yield a compile-time error because the compiler will detect that v.get(0) returns String instead of Integer. For a more elaborate example, see reference.

Here is a small excerpt from the definition of the interfaces List and Iterator in package :

```
public interface List {

void add(E x);
  Iterator iterator();

}

public interface Iterator {

E next();
  boolean hasNext();

}
```

6.4. Type wildcards

A type argument for a parameterized type is not limited to a concrete class or interface. Java allows the use of type wildcards to serve as type arguments for parameterized types. Wildcards are type arguments in the form ""; optionally with an upper or lower bound. Given that the exact type represented by a wildcard is unknown, restrictions are placed on the type of methods that may be called on an object that uses parameterized types.

Here is an example where the element type of a Collection is parameterized by a wildcard:

```
Collection c   new ArrayList();
c.add(new Object()); // compile-time error
c.add(null); // allowed
```

Since we don’t know what the element type of c stands for, we cannot add objects to it. The add() method takes arguments of type E, the element type of the Collection generic interface. When the actual type argument is ?, it stands for some unknown type. Any method argument value we pass to the add() method would have to be a subtype of this unknown type. Since we don't know what type that is, we cannot pass anything in. The sole exception is null; which is a member of every type.

To specify the upper bound of a type wildcard, the extends keyword is used to indicate that the type argument is a subtype of the bounding class. So List<? extends Number> means that

the given list contains objects of some unknown type which extends the Number class. For example, the list could be List<Float> or List<Number>. Reading an element from the list will return a Number. Adding null elements is, again, also allowed.TEMPLATE[Cite_web, url https://docs.oracle.com/javase/tutorial/extra/generics/wildcards.html, title Wildcards > Bonus > Generics, last Bracha, first Gilad, author-link Gilad Bracha, publisher Oracle, website The Java™ Tutorials, quote ...The sole exception is null, which is a member of every type...]

The use of wildcards above adds flexibility since there is not any inheritance relationship between any two parameterized types with concrete type as type argument. Neither List nor List is a subtype of the other; even though Integer is a subtype of Number. So, any method that takes List as a parameter does not accept an argument of List. If it did, it would be possible to insert a Number that is not an Integer into it; which violates type safety. Here is an example that demonstrates how type safety would be violated if List were a subtype of List:

```
List ints   new ArrayList();
ints.add(2);
List nums   ints;  // valid if List were a subtype of List according to substitution rule.
nums.add(3.14);
Integer x   ints.get(1); // now 3.14 is assigned to an Integer variable!
```

The solution with wildcards works because it disallows operations that would violate type safety:

```
List nums   ints;  // OK
nums.add(3.14); // compile-time error
nums.add(null); // allowed
```

To specify the lower bounding class of a type wildcard, the super keyword is used. This keyword indicates that the type argument is a supertype of the bounding class. So, List<? super Number> could represent List<Number> or List<Object>. Reading from a list defined as List<? super Number> returns elements of type Object. Adding to such a list requires either elements of type Number, any subtype of Number or null (which is a member of every type).

The mnemonic PECS (Producer Extends, Consumer Super) from the book Effective Java by Joshua Bloch gives an easy way to remember when to use wildcards (corresponding to covariance and contravariance) in Java.

6.5. Generic class definitions

Here is an example of a generic Java class, which can be used to represent individual entries (key to value mappings) in a map:

```
public class Entry {

private final KeyType key;
  private final ValueType value;

  public Entry(KeyType key, ValueType value) {
     this.key   key;
     this.value   value;
  }

  public KeyType getKey() {
     return key;
  }

  public ValueType getValue() {
     return value;
  }

  public String toString() {
     return "(" + key + ", " + value + ")";
  }

}
```

This generic class could be used in the following ways, for example:

```
Entry grade   new Entry("Mike", "A");
Entry mark   new Entry("Mike", 100);
System.out.println("grade: " + grade);
System.out.println("mark: " + mark);

Entry prime   new Entry(13, true);
if (prime.getValue()) System.out.println(prime.getKey() + " is prime.");
else System.out.println(prime.getKey() + " is not prime.");
```

It outputs:

grade: (Mike, A)
mark: (Mike, 100)
13 is prime.

6.6. Diamond operator

Java SE 7 and above allow the programmer to substitute an empty pair of angle brackets (<>, called the diamond operator) for a pair of angle brackets containing the one or more type parameters that a sufficiently-close context implies.http://docs.oracle.com/javase/7/docs/technotes/guides/language/type-inference-generic-instance-creation.html Thus, the above code example using Entry can be rewritten as:

```
Entry grade   new Entry("Mike", "A");
Entry mark   new Entry("Mike", 100);
System.out.println("grade: " + grade);
System.out.println("mark: " + mark);

Entry prime   new Entry(13, true);
if (prime.getValue()) System.out.println(prime.getKey() + " is prime.");
else System.out.println(prime.getKey() + " is not prime.");
```

6.7. Generic method definitions

Here is an example of a generic method using the generic class above:

```
public static  Entry twice(Type value) {

return new Entry(value, value);

}
```

Note: If we remove the first in the above method, we will get compilation error (cannot find symbol 'Type') since it represents the declaration of the symbol.

In many cases the user of the method need not indicate the type parameters, as they can be inferred:

Entry pair   Entry.twice("Hello");

The parameters can be explicitly added if needed:

Entry pair   Entry.twice("Hello");

The use of primitive types is not allowed, and boxed versions must be used instead:

Entry pair; // Fails compilation. Use Integer instead.

There is also the possibility to create generic methods based on given parameters.

public Type toArray(Type... elements) {

return elements;

}

In such cases you can't use primitive types either, e.g.:

Integer array   toArray(1, 2, 3, 4, 5, 6);

6.8. Generics in throws clause

Although exceptions themselves cannot be generic, generic parameters can appear in a throws clause:

public void throwMeConditional(boolean conditional, T exception) throws T {

if(conditional) {
      throw exception;
  }

}

6.9. Problems with type erasure

Generics are checked at compile-time for type-correctness. The generic type information is then removed in a process called type erasure. For example, List<Integer> will be converted to the non-generic type List, which ordinarily contains arbitrary objects. The compile-time check guarantees that the resulting code is type-correct.

Because of type erasure, type parameters cannot be determined at run-time. For example, when an ArrayList is examined at runtime, there is no general way to determine whether, before type erasure, it was an ArrayList<Integer> or an ArrayList<Float>. Many people are dissatisfied with this restriction. There are partial approaches. For example, individual elements may be examined to determine the type they belong to; for example, if an ArrayList contains an Integer, that ArrayList may have been parameterized with Integer (however, it may have been parameterized with any parent of Integer, such as Number or Object).

Demonstrating this point, the following code outputs "Equal":

```
ArrayList li   new ArrayList();
ArrayList lf   new ArrayList();
if (li.getClass()    lf.getClass()) { // evaluates to true

System.out.println("Equal");

}
```

Another effect of type erasure is that a generic class cannot extend the Throwable class in any way, directly or indirectly:

```
public class GenericException extends Exception
```

The reason why this is not supported is due to type erasure:

```
try {

throw new GenericException();

}
catch(GenericException e) {

System.err.println("Integer");
```

```
}
catch(GenericException e) {

System.err.println("String");

}
```

Due to type erasure, the runtime will not know which catch block to execute, so this is prohibited by the compiler.

Java generics differ from C++ templates. Java generics generate only one compiled version of a generic class or function regardless of the number of parameterizing types used. Furthermore, the Java run-time environment does not need to know which parameterized type is used because the type information is validated at compile-time and is not included in the compiled code. Consequently, instantiating a Java class of a parameterized type is impossible because instantiation requires a call to a constructor, which is unavailable if the type is unknown.

For example, the following code cannot be compiled:

```
T instantiateElementType(List arg) {

return new T(); //causes a compile error

}
```

Because there is only one copy per generic class at runtime, static variables are shared among all the instances of the class, regardless of their type parameter. Consequently, the type parameter cannot be used in the declaration of static variables or in static methods.

6.10. Project Valhalla

Project Valhalla is an experimental project to incubate improved Java generics and language features, for future versions potentially from Java 10 onwards. Potential enhancements include:

1. generic specialization, e.g. List
2. reified generics; making actual types available at runtime.