
UNIT 4 INTRODUCTION TO OBJECT-ORIENTED LANGUAGES

Structure	Page No.
4.0 Introduction	38
4.1 Objectives	39
4.2 Objective-C	39
4.3 Python	40
4.4 C# (C Sharp)	41
4.5 Eiffel	42
4.6 Modula-3	43
4.7 Small Talk	44
4.8 Object REXX	45
4.9 JAVA	46
4.10 BETA	48
4.11 Various Object-Oriented Programming Languages Comparative Chart	50
4.12 Summary	51
4.13 Solutions/Answers	51

4.0 INTRODUCTION

Object-oriented programming offers a new and powerful model for writing computer software. Object-oriented programming, like most interesting developments, builds on some old ideas, extends them, and puts them together in novel ways. The result is many faceted and a clear step forward for the art of programming. An object-oriented approach makes programs more intuitive to design, faster to develop, more amenable to modifications, and easier to understand. It leads not only to alternative ways of constructing programs, but also to alternative ways of conceiving the programming task.

Object-Orientation (OO), or to be more precise, object-oriented programming, is a problem-solving method in which the software solution reflects objects in the real world. Objects are “black boxes” which send and receive messages. This approach speeds the development of new programs, and, if properly used, improves the maintenance, reusability, and modifiability of software.

The C++ language offers an easier transition via C, but it still requires an object-oriented design approach in order to make proper use of this technology. Smalltalk offers a pure object-oriented environment, with more rapid development time and greater flexibility and power. Java promises much for Web-enabling object-oriented programs.

There are almost two dozen major object-oriented programming languages in use today. But the leading commercial object-oriented languages are far fewer in number. In this unit, let us look into the features of the mostly used object-oriented programming languages. Some of the details provided in this unit are implementation oriented. Therefore, it is advisable that you may read the unit again after having practical experiences.

4.1 OBJECTIVES

After going through this Unit, you will be able to:

- describe the importance of object oriented programming languages, and
 - discuss the features of various object oriented programming languages.
-

4.2 OBJECTIVE - C

C is a low-level, block-structured language often used for systems programming. C++ is an object-oriented (OO) language developed as an extension of C and used for application development.

Objective C is an object-oriented programming language based on C and used for application and library programming. The Objective-C language is fully compatible with ANSI standard C. Objective-C was invented by Brad Cox who wrote the book “Object Oriented Programming: An Evolutionary Approach” in which he describes the language.

Features of Objective-C

Objective-C includes, when compared to C, a few more keywords and constructs. Objective-C is a powerful, easy-to-learn, object-oriented extension to C. Advanced object-oriented features like dynamic binding, run-time type identification, and persistence are standard features in Objective-C which apply universally and work well together. Moreover, the support for descriptive message names (as in SmallTalk) makes Objective-C code easy to read and understand. The GNU and NeXTSTEP C compilers support objective-C.

- Objective-C is a superset of the C programming language, and may be used to develop non-OO and OO programs. Objective-C provides access to scalar types, structures and to unions. Objective-C provides zero-cost access to existing software libraries written in C.
- Objective-C is dynamically typed but also provides static typing.
- Objective-C has no such conventions or tool support in place.
- Objective-C provides delegation (the benefits of multiple inheritance without the drawbacks) at minimal programming cost.
- Objective-C provides manual memory management, reference counting, and garbage collection as options.
- Objective-C provides protocols.
- Objective-C has classes similar to Smalltalk. Objective-C is as close to Smalltalk as a compiled language allows.
- Objective-C is compiled.
- You may add or delete methods and classes at runtime.
- Objective-C has meta classes.
- Objective-C does not have class variables, but it has pool variables and globals which are easily emulated via static variables.
- The possibility to load class definitions and method definitions (which extend a class) at run time.
- Objects are dynamically typed.
- Persistence is there in Objective-C.
- Remote objects are possible.
- It allows Delegation and target/action protocols.
- There is no innate multiple inheritance.

- No class variables.

4.3 PYTHON

Python is a language that has always aimed at consistency, simplicity, ease of understanding, and portability. It is simple, object-oriented, extensible, robust, scalable and features an easy to learn syntax that is clear and concise. Python combines the power of a compiled object language like Java and C++ with the ease of use. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and Rapid Application Development (RAD) in many areas on most platforms.

The python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Website, <http://www.python.org/> and can be freely distributed. The same site also contains distributions of pointers to many free third party Python modules, programs and tools, and additional documentation. Let us see the features of the Python language in the next section.

Features of Python

- The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.
- It offers much more error checking than C, and, being a very high level language, it has high-level data types built in, such as flexible arrays and dictionaries that would cost you days to implement efficiently in C. Because of its more general data types, Python is applicable to a much larger problem domain than Awk or even Perl, yet many things are at least as easy in Python as in those languages.
- Python allows you to split up your program in modules that can be reused in other Python programs. It comes with a large collection of standard modules that you can use as the basis of your programs - or as examples to start learning to program in Python. There are also built-in modules that provide things like file I/O, system calls, sockets, and even interfaces to graphical user interface toolkits like Tk.
- Python is an interpreted language, which can save you considerable time during program development because no compilation and linking is necessary. The interpreter can be used interactively, which makes it easy to experiment with features of the language, to write throw-away programs, or to test functions during bottom-up program development.
 - the high-level data types allow you to express complex operations in a single statement;
 - statement grouping is done by indentation instead of begin/end brackets;
 - no variable or argument declarations are necessary.
- Python is extensible: If you know how to program in C, it is easy to add a new built-in function or module to the interpreter, either to perform critical operations at maximum speed, or to link Python programs to libraries that

may only be available in binary form (such as a vendor-specific graphics library).

- TCP/IP and UDP/IP Network programming using sockets
 - Operating system interface is possible
 - GUI development with Tk using Tkinter
 - It allows multithreaded programming
 - We can build interactive Web/CGI/Internet applications
 - Inheritance, type emulation, operator overloading, and delegation in an OOP environment.
-

4.4 C# (C SHARP)

C# (C Sharp) is a modern, object-oriented language that enables programmers to quickly build a wide range of applications for the new Microsoft.NET platform, which provides tools and services that fully exploit both computing and communications. This is the premier language for the Next Generation Windows Services (NGWS) Runtime. This NGWS runtime is a runtime environment that not only manages the execution of code, but also provides services that make programming easier. Compilers produce managed code to target this managed execution environment. You get cross-language integration, cross-language exception handling, enhanced security, versioning and deployment support, and debugging and profiling services for free. C# derived from C and C++, however it is modern, simple, entirely object oriented and type safe.

Because of its elegant object-oriented design, C# is a great choice for architecting a wide range of components-from high-level business objects to system-level applications. Using simple C# language constructs, these components can be converted into Extensible Markup Language (XML) Web services, allowing them to be invoked across the Internet, from any language running on any operating system.

More than anything else, C# is designed to bring rapid development to the C++ programmer without sacrificing the power and control that have been a hallmark of C and C++. Because of this heritage, C# has a high degree of fidelity with C and C++. Developers familiar with these languages can quickly become productive in C#. As C# is a modern programming language, it simplifies and modernizes C# in the areas of classes, namespaces, method overloading, and exceptional handling. Contributing to the ease of use is the elimination of certain features of C++: no more macros, no templates, and no multiple inheritance and pointers.

Features of C#

- **Simple**

C# is a simple language. Pointers are missing in C#. In C++, we have ::, . (dot) and -> operators that are used for namespaces, members and references. C# provides a unified type system. This type system enables you to view every type as an object, be it a primitive data type or a full-blown class.

- **Modern**

C# is designed to be the premier language for writing NGWS applications. The entire memory management is no longer the duty of the programmer – the runtime of NGWS provides a garbage collector that is responsible for memory management in the C# programs. Exception handling is cross-language (another feature of runtime). It provides you metadata syntax for declaring capabilities and permissions for the underlying NGWS security model.

- **Object-Oriented**

C# supports all the key object-oriented concepts such as encapsulation, inheritance, and polymorphism. The entire C# class model is built on top of the NGWS runtime's Virtual Object System (VOS). Accidental overriding of methods is overcome in C#. C# supports the private, protected, public and internal access modifiers. C# allows only one base class. If the programmer needs the feel for multiple inheritance, he can implement interface.

- **Type-safe**

We cannot use uninitialized variables. For member variables of an object, the compiler takes care of zeroing them. For local variables, the programmer should take care of. C# does away with unsafe casts. A bound checking is part of C#. Arithmetic operations could overflow the range of the result data-type. C# allows you to check for overflow in such operations on either an application level or a statement level. Reference parameters that are passed in C# are type-safe.

- **Versionable**

In C#, the versioning support for applications is provided by the NGWS runtime. C# does its best to support this versioning. Although C# itself cannot guarantee correct versioning, it can ensure that versioning is possible for the programmer.

- **Compatible**

C# allows you to access to different API's with the foremost being the NGWS common Language Specification (CLS). The CLS defines a standard for interoperability between languages that adhere to this standard. You can also access the older COM objects. C# supports the OLE automation. Finally, C# enables you to interoperate with C-style API's.

- **Flexible**

If you need pointers, you can still use them via unsafe code - and no marshalling is involved when calling the unsafe code.

4.5 EIFFEL

Eiffel is a pure object-oriented language created by Bertrand Meyer and developed by his company, Interactive Software Engineering (ISE) of Goleta, Canada. The language was introduced in 1986. Eiffel is named after Gustave Eiffel, the engineer who designed the Eiffel Tower.

Eiffel encourages object oriented programming development and supports a systematic approach to software development. Eiffel has an elegant design and programming style, and is easy to learn.

The Eiffel compiler generates C code, which you can then modify and re-compile with a C compiler. Its modularity is based on classes. It stresses reliability, and facilitates design by contract. It brings design and programming closer together. It encourages the re-use of software components. Eiffel offers classes, multiple inheritance, polymorphism, static typing and dynamic binding, genericity (constrained and unconstrained), a disciplined exception mechanism, systematic use of assertions to promote programming by contract, and deferred classes for high-level design and analysis.

It is hard to generalise, but compared to C++, simple computation-intensive applications will run perhaps 15% slower. Large applications are often dominated by memory management rather than computation. ISE recently demonstrated that by simply adding a call to the garbage collector's "full-collect" routine at a time when there were known to be few live objects, performance became dramatically faster than a corresponding C++ version.

Features of Eiffel

- **Portable:** This language is available for major industry platforms, such as Windows, OS/2, Linux, UNIX, VMS, etc..
- **Open System** includes a C and C++ interface making it easily possible to reuse code previously written.
- **"Melting Ice Technology"** combines compilation, for the generation of efficient code, with bytecode interpretation, for fast turnaround after a change.
- **"Design by Contract"** enforced through assertions such as class invariants, preconditions and postconditions.
- **Automatic Documentation ("Short Form"):** Abstract yet precise documentation produced by the environment at the click of a button.
- **Multiple Inheritance:** A class can inherit from as many parents as necessary.
- **Repeated Inheritance:** A class inherits from another through two or more parents.
- **Statically Typed** ensure that errors are caught at compile time, rather than run time.
- **Dynamically Bound** guarantees that the right version of an operation will always be applied depending on the target object.
- **The Few Interfaces Principle** restricts the overall number of communication channels between modules in a software architecture:
"Every module should communicate with as few others as possible".

4.6 MODULA-3

Modula-3 programming language is from Digital Equipment Corporation's Systems Research Center (SRC). Modula-3 is a modern, modular, object-oriented language. One of the principal goals for the Modula-3 language was to be simple and comprehensible, yet suitable for building large, robust, long-lived applications and systems. The language design process was one of consolidation and not innovation;

that is, the goal was to consolidate ideas from several different languages, ideas that had proven useful for building large sophisticated systems.

The language features garbage collection, exception handling, run-time typing, generics, and support for multithreaded applications. The SRC implementation of this language features a native-code compiler; an incremental, generational, conservative, multithreaded garbage collector; a minimal recompilation system; a debugger; a rich set of libraries; support for building distributed applications; a distributed object-oriented scripting language; and finally; a graphical user interface builder for distributed applications.

Features of MODULA-3

- Modula-3 has a particularly simple definition of an object. In Modula-3, an object is a record on the heap with an associated method suite. The data fields of the object define the state, and the method suite defines the behaviour. The Modula-3 language allows the state of an object to be hidden in an implementation module with only the behaviour visible in the interface. This is different from C++ where a class definition lists both the member data and member function. The C++ model reveals what is essentially private information (namely the state) to the entire world. With Modula-3 objects, what should be private can really be private.
- Modula-3 has the feature of garbage collection. Garbage collection really enables robust, long-lived systems.
- Modula-3 provides such a standard interface for creating threads. In addition, the language itself includes support for managing locks. The standard libraries have provided an interface to X, is not only thread-safe, but uses threads to carry out long operations in the background.
- Generic interfaces and modules are a key to reuse. Modula-3 generics are cleaner than C++ parameterized types, but provide much of the same flexibility.
- Modula-3 provides a simple single-inheritance object system.
- Existing non-Modula-3 libraries can be imported. Many existing C libraries make extensive use of machine-dependent operations. These can be imported as “unsafe”. Then, safer interfaces can be built on top of these while still allowing access to the unsafe features of the libraries for those applications that need them.

4.7 SMALL TALK

Smalltalk is a purely object-oriented language which cleanly supports the notion of classes, methods, messages and inheritance. Smalltalk is a programming language developed in the 1970s at Xerox’s Palo Alto Research Center in California.

Unlike “hybrid” object-oriented languages C++, Smalltalk is considered to be a “pure” object-oriented language. Smalltalk is said to be “pure” for one main reason: Everything in Smalltalk is an object, whereas in hybrid systems there are things, which are not objects (for example, integers in C++ and java).

Smalltalk is fundamentally tied to automatic dynamic memory management, and as such must be supported by an underlying automatic memory management system. In practical terms, this means that a Smalltalk programmer no longer needs to worry

about when to free allocated memory. When finished with a dynamically allocated object, the program can simply “walk away” from the object. The object is automatically freed, and its storage space recycled, when there is nothing else referencing it.

All Smalltalk code is composed of chains of messages sent to objects. Even the programming environment itself is designed within this metaphor. A large number of predefined classes are collectively responsible for the system’s impressive functionality. Different from most other programming tools all of this functionality is always accessible to browsing and change, a fact which makes Smalltalk an extremely flexible system, which is easy to customize according to one’s own preferences. Smalltalk programs are considered by most to be significantly faster to develop. A rich class library that can be easily reused via inheritance is one reason for this. Smalltalk has development environment is more dynamic in nature. It is not explicitly compiled. This makes the development process definitions easily refined. The various versions of Smalltalk language are given as follows:

Open Source & Free Smalltalk Versions

- Squeak Smalltalk
- GNU Smalltalk
- Little Smalltalk

Commercial Smalltalk Versions

- Dolphin Smalltalk
- Object Connects Smalltalk MT
- Expert’s Smalltalk/X
- Cincom’s Visual Works Smalltalk
- Cincom’s Object Studio Smalltalk
- IBM’s Visual Age Smalltalk
- Pocket Smalltalk
- QKS Smalltalk Agents

4.8 OBJECT TEXX

IBM Object REXX is an object-oriented programming language suited for beginners as well as experienced OO programmers. It is upward compatible with previous versions of classic REXX and provides an easy migration path to the world of objects. Object REXX runs on the operating systems like AIX, Linux, OS/2, Windows 2000, Windows 98, Windows Me and Windows NT. The following are the features of REXX:

Features of REXX

- Suitable for solving small automation problems and developing fully realized applications.
- Includes a rich set of system interfaces.
- Can be used for writing powerful command procedures for Windows.
- Includes the complete Object REXX Interpreter.
- Runs immediately without compilation or linkage.
- Can develop and debug Object REXX applications, including GUIs (Development Edition)
- Support for OLE/Active X.
- UNICODE conversion functions.

- Enhanced with full object orientation.
 - Designed for object-oriented programming, and also allows REXX conventional programming.
 - Provides a REXX API to develop external function libraries written in C.
 - Includes applications developed in C or C++.
-

4.9 JAVA

Java was developed by taking the best points from other programming languages, primarily C and C++. Java therefore utilises algorithms and methodologies that are already proven. Error prone tasks such as pointers and memory management have either been eliminated or are handled by the Java environment automatically rather than by the programmer. Since Java is primarily a derivative of C++ which most programmers are conversant with, it implies that Java has a familiar feel rendering it easy to use.

Features of Java

- **Object-oriented**

Java's basic execution unit is the class. Advantages of OOP include: reusability of code, extensibility and dynamic applications.

- **Distributed**

Commonly used Internet protocols such as HTTP and FTP as well as calls for network access are built into Java. Internet programmers call on the functions through the supplied libraries and be able to access files on the Internet as easily as writing to a local file system.

- **Interpreted**

When Java code is compiled, the compiler outputs the Java Bytecode which is an executable for the Java Virtual Machine. The Java Virtual Machine does not exist physically, but is the specification for a hypothetical processor that can run Java code. The bytecode is then run through a Java interpreter on any given platform that has the interpreter ported to it. The interpreter converts the code to the target hardware and executes it.

- **Robust**

Java compels the programmer to be thorough. It carries out type checking at both compile and runtime making sure that every data structure has been clearly defined and typed. Java manages memory automatically by using an automatic garbage collector. The garbage collector runs as a low priority thread in the background keeping track of all objects and references to those objects in a Java program. When an object has no more references, the garbage collector tags it for removal and removes the object either when there is an immediate need for more memory or when the demand on processor cycles by the program is low.

- **Secure**

The Java language has built-in capabilities to ensure that violations of security do not occur. Consider a Java program running on a workstation on a local area network which in turn is connected to the Internet. Being a dynamic and distributed computing

environment, the Java program can, at runtime, dynamically bring in the classes it needs to run either from the workstation's hard drive, other computers on the local area network or a computer thousands of miles away somewhere on the Internet. This ability of classes or applets to come from unknown locations and execute automatically on a local computer sounds like every system administrator's nightmare considering that there could be lurking out there on one of the millions of computers on the Internet some viruses, trojan horses or worms which can invade the local computer system and wreak havoc on it.

Java goes to great lengths to address these security issues by putting in place a very rigorous multilevel system of security:

- First and foremost, at compile time, pointers and memory allocation are removed thereby eliminating the tools that a system breaker could use to gain access to system resources. Memory allocation is deferred until runtime.
- Even though the Java compiler produces only correct Java code, there is still the possibility of the code being tampered with between compilation and runtime. Java guards against this by using the bytecode verifier to check the bytecode for language compliance, when the code first enters the interpreter, before it ever gets the chance to run.
- Even though the Java compiler produces only correct Java code, there is still the possibility of the code being tampered with between compilation and runtime. Java guards against this by using the bytecode verifier to check the bytecode for language compliance when the code first enters the interpreter, before it even gets the chance to run.

The bytecode verifier ensures that the code does not do any of the following:

- Forge pointers
- Violated the access restrictions
- Incorrectly access classes
- Overflow or underflow operand stack
- Use incorrect parameter of bytecode instructions
- Use illegal data conversions.
- At runtime, the Java interpreter further ensures that classes loaded do not access the file system except in the manner permitted by the client or the user.
- **Architecturally neutral**

The Java compiler compiles source code to a stage which is intermediate between source and native machine code. This intermediate stage is known as the bytecode, which is neutral. The bytecode conforms to the specification of a hypothetical machine called the Java Virtual Machine and can be efficiently converted into native code for a particular processor.

- **Portable**

By porting an interpreter for the Java Virtual Machine to any computer hardware/operating system, one is assured that all codes compiled for it will run on that system. This forms the basis for Java's portability.

Another feature which Java employs in order to guarantee portability is by creating a single standard for data sizes irrespective of processor or operating system platforms.

- **High performance**

The Java language supports many high-performance features such as **multithreading, just-in-time compiling, and native code usage**.

- Java has employed multithreading to help overcome the performance problems suffered by interpreted code as compared to native code. Since an executing program hardly ever uses **CPU** cycles 100% of the time, Java uses the idle time to perform the necessary garbage cleanup and general system maintenance that renders traditional interpreters slow in executing applications. [NB: Multithreading is the ability of an application to execute more than one task (thread) at the same time, e.g., a word processor can be carrying out spell check in one document and printing a second document at the same time.]
- Since the bytecode produced by the Java compiler from the corresponding source code is very close to machine code, it can be interpreted very efficiently on any platform. In cases where even greater performance is necessary than the interpreter can provide, just-in-time compilation can be employed where by the code is compiled at run-time to native code before execution.
- An alternative to just-in-time compilation is to link in native **C** code. This yields even greater performance, but is more burdensome on the programmer and reduces the portability of the code.
- **Dynamic**

By connecting to the Internet, a user immediately has access to thousands programs and other computers. During the execution of a program, Java can dynamically load classes that it requires either from the local hard drive, from another computer on the local area network or from a computer somewhere on the Internet.

4.10 BETA

BETA is a modern object-oriented language with comprehensive facilities for procedural and functional programming . BETA originates from the Scandinavian school of object-orientation where the first object-oriented language Simula was developed. Object-oriented programming originated with the Simula languages developed at the Norwegian Computing Center, Oslo, in the 1960s.

The BETA language development process started out in 1975 to develop concepts, constructs and tools for programming, partly based on the Simula languages. The BETA language team consists of Bent Bruun Kristensen, Birger Moller-Predersen, Ole Lehramann Madsen, and Kristen Nygaard. Kristen Nygaard was one of the two original designers of the Simula languages. Currently, BETA is available on UNIX workstations, on PowerPC Macintosh and on Intel-based PCs.

On UNIX, the platforms supported are: Sun Sparc (Solaris), HP 9000 (series 7000 and Silicon Graphics MIPS machines running IRIX 5.3 or 6.

The definition of the BETA language is in the public domain. This definition is controlled by the original designers of the BETA language: Bent Bruun Kristensen,

Ole Lehrmann Madsen, Birger Moller-Pedersen, and Kristen Nygarrd. This means that anyone or any company may create a compiler, interpreter, or whatever having to do with BETA.

BETA has powerful abstraction mechanisms. It provides excellent support for design and implementation, including data definition for persistent data. The abstraction mechanisms include support for identification of objects, classification, and composition. BETA is a strongly typed language with most type checking being carried out a compile-time.

The abstraction mechanisms include class, procedure, function, coroutine, process, exception, and many more, all unified into the ultimate abstraction mechanism: the pattern. In addition to the pattern, BETA has subpattern, virtual pattern, and pattern variable.

BETA does not only allow for passive objects as in Smalltalk, C++, and Eiffel. BETA objects may also act as co-routines, making it possible to model alternating sequential processes and quasi-parallel processes. BETA co-routines may also be executed concurrently with supported facilities for synchronization and communication, including monitors and rendezvous communication.

Features of BETA

BETA replaces classes, procedures, functions, and types by a single abstraction mechanism, called the pattern. It generalizes virtual procedures, virtual patterns, streamlines linguistic notions such as nesting and block structure, and provides a unified framework for sequential, co-routine, and concurrent execution.

The pattern concept is the basic construct. A pattern is a description from which objects may be created. Patterns describe all aspects of objects, such as attributes and operations, as seen in traditional object-oriented languages, but also aspects such as parameters and actions, as seen in procedures.

Objects are created from the patterns. Objects may be traditional objects as found in other languages, but they may also be objects which correspond to procedure or function activations, exception occurrences, or even co-routines or concurrent processes.

Objects may be created statically or dynamically and the objects are automatically garbage-collected by the runtime system when no references exist to them any longer.

Patterns may be used as super patterns to other patterns (the sub-patterns). This corresponds to traditional class hierarchies, but since patterns may describe other types of objects, inheritance is a structuring means available also for procedures, functions, exceptions, co-routines and processes.

Patterns may be virtual. This corresponds to traditional virtual procedures but again the generality of the pattern construct implies that also classes, exceptions, co-routines, and processes may be virtual.

Virtual patterns in the form of classes are similar to generic templates in other languages. The prime difference is that generic parameters (that is, the virtual class patterns) may be further restricted without actually instantiating the generic template. The generality of the pattern also implies that generativity is available for classes, procedures, functions, exceptions, coroutines, and processes.

Patterns may be handled as first-order values in BETA. This implies the possibility of defining pattern variables which can be assigned pattern references dynamically at

runtime. This gives the possibilities for a very dynamic handling of patterns at runtime.

Exception handling is dealt with through a predefined library containing basic exception handling facilities. The exception handling facilities are fully implemented within the standard BETA language in the form of a library pattern, and the usage is often in the form of virtual patterns, inheriting from this library pattern.

Garbage collection is conducted automatically by the BETA runtime system when it is discovered that no references to the object exist. The garbage collection mechanism is based on generation-based scavenging. The implemented garbage collection system is very efficient.

4.11 VARIOUS OBJECT-ORIENTED PROGRAMMING LANGUAGES COMPARATIVE CHART

In the earlier sections we had gone through the features of various object oriented programming languages. The following chart provides the comparison of the features of the languages:

Language	BETA	C++	Eiffel	Java	Object Pascal	Ruby	Smalltalk
Inheritance	simple	multiple	multiple	simple	simple	simple	simple
Generic Classes (templates)	yes	yes	yes	no	no	no	no
Strong typing	yes	yes	yes	yes	yes	no	no
Polymorphic	yes	yes	yes	yes	yes	yes	yes
Multithreading	yes	possible	possible	yes	possible	possible	possible
Garbage collection	yes	no	yes	yes	no	yes	yes
Pre-/ post-conditions	indirect (through patterns)	no	yes	no	no	no	no
Speed	+	+++	++	--	++	--	--
Hybrid/OOP	hybrid (pattern)	hybrid	OOP	OOP	hybrid	OOP	OOP
Compiler/interpreter	compiler	compiler	compiler	interpreter	compiler	interpreter	interpreter
Special environment	browser/pretty-printer	-	browser/pretty printer	-	-	-	class browser
Special concept	pattern	-	designed by contract (Bertrand Meyer)	-	-	everything is an object	everything is an object

Check Your Progress

- 1) What is Object-Oriented Programming?

.....
.....

- 2) What are Object-Oriented Tools?

.....
.....

- 3) What is Object-Oriented Analysis and Design?

.....
.....

4.12 SUMMARY

Object-orientation is a new programming concept, which should help you in developing high quality software. Object-orientation makes developing of projects easier. Complex software systems become easier to understand, since object-oriented structuring provides a closer representation of reality than other object-oriented programming techniques. In a well-designed object-oriented system, it should be possible to implement changes at class level, without having to make alterations at other points in the system. This reduces the overall amount of maintenance required. Through polymorphism and inheritance, object-oriented programming allows you to reuse individual components. In an object-oriented system, the amount of work involved in revising and maintaining the system is reduced since many problems can be detected and corrected in the design phase. This unit has presented an overview of some of the important object-oriented languages.

4.13 SOLUTIONS/ANSWERS

Check Your Progress

- 1) A type of programming in which programmers define not only the data type of data structure, but also the types of operations (functions) that can be applied to the data structure. In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects.

One of the principal advantages of object-oriented programming techniques over procedural programming techniques is that they enable programmers to create modules that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify.

- 2) Object-oriented tools allow you to create object-oriented programs in object-oriented languages. They allow you to model and store development objects and the relationships between them.
- 3) OO Analysis – Examination of requirements from the perspective of the classes/objects found in the problem domain.

OO Design–Uses OO decomposition and a notation for depicting logical/physical and static/dynamic models of the system.

An Introduction to Object-Oriented Programming