DOWNLOAD PDF OBJECT ORIENTED PROGRAMMING CONCEPTS BOOK

Chapter 1 : Introduction to Object Oriented Programming Concepts (OOP) and More - CodeProject

of over 8, results for Books: "Object-Oriented Programming" The Fundamental Concepts of Object-Oriented Programming Oct 7, by Dimitrios Kalemis.

OOP is a design philosophy. It stands for Object Oriented Programming. Object-Oriented Programming OOP uses a different set of programming languages than old procedural programming languages C, Pascal, etc. Everything in OOP is grouped as self sustainable "objects". Hence, you gain reusability by means of four main object-oriented programming concepts. Your body has two objects of the type "hand", named "left hand" and "right hand". Their main functions are controlled or managed by a set of electrical signals sent through your shoulders through an interface. So the shoulder is an interface that your body uses to interact with your hands. The hand is a well-architected class. The hand is being reused to create the left hand and the right hand by slightly changing the properties of it. What is an Object? An object can be considered a "thing" that can perform a set of related activities. For example, the Hand object can grip something, or a Student object can give their name or address. In pure OOP terms an object is an instance of a class. What is a Class? A class is simply a representation of a type of object. A class is the blueprint from which the individual objects are created. Class is composed of three things: As an example, there may be thousands of other bicycles in existence, all of the same make and model. Each bicycle has built from the same blueprint. In object-oriented terms, we say that the bicycle is an instance of the class of objects known as bicycles. In the software world, though you may not have realized it, you have already used classes. For example, the TextBox control, you always used, is made out of the TextBox class, which defines its appearance and capabilities. Each time you drag a TextBox control, you are actually creating a new instance of the TextBox class. How to identify and design a Class? This is an art; each designer uses different techniques to identify classes. However according to Object Oriented Design Principles, there are five principles that you must follow when design a class, SRP - The Single Responsibility Principle - A class should have one, and only one, reason to change. For more information on design principles, please refer to Object Mentor. Additionally to identify a class correctly, you need to identify the full list of leaf-level functions or operations of the system granular level use cases of the system. In software world the concept of dividing and conquering is always recommended, if you start analyzing a full system at the start, you will find it harder to manage. So the better approach is to identify the module of the system first and then dig deep in to each module separately to seek out classes. A software system may consist of many classes. When you have many classes, it needs to be managed. In order to manage such a work force, you need to have proper management policies in place. Same technique can be applied to manage classes of your software system. In order to manage the classes of a software system, and to reduce the complexity, system designers use several techniques, which can be grouped under four main concepts named 1. What is Encapsulation or Information Hiding? The encapsulation is the inclusion-within a program object-of all the resources needed for the object to function, basically, the methods and the data. In OOP the encapsulation is mainly achieved by creating classes, the classes expose public methods and properties. A class is kind of a container or capsule or a cell, which encapsulate a set of methods, attribute and properties to provide its indented functionalities to other classes. In that sense, encapsulation also allows a class to change its internal implementation without hurting the overall functioning of the system. That idea of encapsulation is to hide how a class does its business, while allowing other classes to make requests of it. According to Object Oriented Programming there are several techniques classes can use to link with each other. There are several other ways that an encapsulation can be used, as an example we can take the usage of an interface. The interface can be used to hide the information of an implemented class. It allows one object instance to cause another to perform an action on its behalf. Association is the more general term that define the relationship between two classes, where as the aggregation and composition are relatively special. Since a direction is explicitly specified, in this case the controller class is the StudentRegistrar. To some beginners, association is a confusing concept. The troubles created not only by the association alone, but with two other OOP concepts, that is association, aggregation and composition. Every one understands association, before aggregation and

composition are described. Aggregation or composition cannot be separately understood. What is the difference between Association, Aggregation, and Composition? It is just the connectivity between the two classes. Aggregation is a weak type of Association with partial ownership. This is weak compared to Composition. Then again, weak meaning the linked components of the aggregator may survive the aggregations life-cycle without the existence of their parent objects. Any teacher may belong to more than one department. And so, if a department ceases to exist, the teacher will still exist. On the other hand, Composition is a strong type of Association with full ownership. Hence, if a department ceases to exist, the underlying courses will cease to exist as well. There is no ownership between the teacher and the student, and each has their own life-cycle. But even without a Chancellor a University can exists. But the Faculties cannot exist without the University, the life time of a Faculty or Faculties attached with the life time of the University. If University is disposed the Faculties will not exist. In that case we called that University is composed of Faculties. Same way, as another example, you can say that, there is a composite relationship in-between a KeyValuePairCollection and a KeyValuePair. I see Composition is being used in many other ways too. However the more important factor, that most people forget is the life time factor. The life time of the two classes that has bond with a composite relation mutually depend on each other. If you take the. NET Collection to understand this, there you have the Collection element define inside it is an inner part, hence called it is composed of the Collection, farcing the Element to get disposed with the Collection. So the point is, if you want to bind two classes with Composite relation, more accurate way is to have a one define inside the other class making it a protected or private class. This way you are allowing the outer class to fulfill its purpose, while tying the lifetime of the inner class with the outer class. So in summary, we can say that aggregation is a special kind of an association and composition is a special kind of an aggregation. What is Abstraction and Generalization? Abstraction is an emphasis on the idea, qualities and properties rather than the particulars a suppression of detail. The importance of abstraction is derived from its ability to hide irrelevant details and from the use of names to reference objects. Abstraction is essential in the construction of programs. It places the emphasis on what an object is or does rather than how it is represented or how it works. Thus, it is the primary means of managing complexity in large programs. While abstraction reduces complexity by hiding irrelevant detail, generalization reduces complexity by replacing multiple entities which perform similar functions with a single construct. Generalization is the broadening of application to encompass a larger domain of objects of the same or different type. Programming languages provide generalization through variables, parameterization, generics and polymorphism. It places the emphasis on the similarities between objects. Thus, it helps to manage complexity by collecting individuals into groups and providing a representative which can be used to specify any individual of the group. Abstraction and generalization are often used together. Abstracts are generalized through parameterization to provide greater utility. In parameterization, one or more parts of an entity are replaced with a name which is new to the entity. The name is used as a parameter. When the parameterized abstract is invoked, it is invoked with a binding of the parameter to an argument. What is an Abstract class? Abstract classes, which declared with the abstract keyword, cannot be instantiated. It can only be used as a super-class for other classes that extend the abstract class. Abstract class is the concept and implementation gets completed when it is being realized by a subclass. Abstract classes are ideal when implementing frameworks. Please carefully read the comments as it will help you to understand the reasoning behind this code. This class will allow all subclass to gain access to a common exception logging module and will facilitate to easily replace the logging library. But you do have a concept in mind and that is, if a class is going to log an exception, they have to inherit the LoggerBase.

Chapter 2 : ABAP Objects: Object-Oriented Programming (OOP) | Book - by SAP PRESS

The Fundamental Concepts of Object-Oriented Programming and millions of other books are available for Amazon Kindle. Learn more Enter your mobile number or email address below and we'll send you a link to download the free Kindle App.

Read on for a primer on OOP concepts in Java. They are an abstraction , encapsulation , inheritance , and polymorphism. Grasping them is key to understanding how Java works. Basically, Java OOP concepts let us create working methods and variables, then re-use all or part of them without compromising security. Abstraction means using simple things to represent complexity. In Java, abstraction means simple things like objects, classes, and variables represent more complex underlying code and data. This is important because it lets avoid repeating the same work multiple times. This is the practice of keeping fields within a class private, then providing access to them via public methods. This way, we can re-use objects like code components or variables without allowing open access to the data system-wide. This is a special feature of Object Oriented Programming in Java. It lets programmers create new classes that share some of the attributes of existing classes. This lets us build on previous work without reinventing the wheel. This Java OOP concept lets programmers use the same word to mean different things in different contexts. One form of polymorphism in Java is method overloading. The other form is method overriding. See more on this below. For example, a programmer can create several different types of objects. These can be variables, functions, or data structures. Programmers can also create different classes of objects. These are ways to define the objects. For instance, a class of variable might be an address. The class might specify that each address object shall have a name, street, city, and zip code. The objects, in this case, might be employee addresses, customer addresses, or supplier addresses. How Encapsulation Works Encapsulation lets us re-use functionality without jeopardizing security. For example, we may create a piece of code that calls specific data from a database. It may be useful to reuse that code with other databases or processes. Encapsulation lets us do that while keeping our original data private. It also lets us alter our original code without breaking it for others who have adopted it in the meantime. It works by letting a new class adopt the properties of another. We call the inheriting class a subclass or a child class. The original class is often called the parent. We use the keyword extends to define a new class that inherits properties from an old class. How Polymorphism Works Polymorphism in Java works by using a reference to a parent class to affect an object in the child class. Two more examples of polymorphism in Java are method overriding and method overloading. In method overriding, the child class can use the OOP polymorphism concept to override a method of its parent class. That is, a single method name might work in different ways depending on what arguments are passed to it. Inheritance can be as easy as using the extends keyword: The Employee class inherits from the Person class by using the keyword extends. Here, the child class overrides the parent class. For the full example, see this blog post. This is the core concept in Java. You should never have two blocks of identical code in two different places. Instead, have one method you use for different applications. If you expect your Java code to change in the future, encapsulate it by making all variables and methods private at the outset. Simply put, a class should always have only one functionality. Make all methods and classes Closed for modification but Open for an extension. That way, tried and tested code can remain static but can be modified to perform new tasks as needed. For a good, full list of best practices for OOP concepts in Java, see this blog post. You can also check out our article about OOP concepts in C. Latest Posts About Stackify Stackify provides developer teams with unparalleled visibility and insight into application health and behavior, both proactively in a monitoring role as well as reactively in a troubleshooting role, while eliminating the need to login to servers and other resources in order to investigate application problems.

Chapter 3 : 6 Great Books for Learning PHP Object-Oriented Programing

Mastering Oriented-Object concepts is so much more than just understand constructs supported by some programming language like class, interface, or calendrierdelascience.com is one of the most important works on the way to becoming a great software developer too.

Overview[edit] Where Procedure-oriented programming uses procedures to make code easier to write and understand, Object-oriented programming OOP goes a step further and uses objects to make code easier to create and work with. In OOP, the word "object" has special meaning: Briefly, an Object is the combination of: The data might include: Procedure-oriented and Object-oriented programming were invented because programs were getting longer and longer, and were difficult to work with. Programmers needed more structure to simplify the programming process. A program of moderate size and complexity can be simplified using procedures. With especially large or complex programs, procedures are not enough; OOP became popular as a way of handling these very complex programs. A program may be complex in the raw source code - many lines of code, many procedures. OOP helps with both. Improvements to Procedure-Oriented Programming[edit] OOP started from observing ways of writing Procedure-oriented programs that were more effective easier to write, had fewer bugs than others. A few techniques that were optional in Procedure-oriented programming became the core of OOP: Aggregation - an aggregate object is an object which contains other objects. Sometimes a class refers to real-world physical objects like a Car. Sometimes it is more abstract e. Composition - composition is like aggregation but it is a real-time technique. Using interfaces you can replace objects at real time. But objects must have the same type. Interfaces - the Interface separates the implementation and defines the structure. This concept is useful when implementation of the object can be interchangeable. Also, please pay attention that you can use this technique if implementation is changing frequently!

Chapter 4 : Object-Oriented Programming - Free Books at EBD

Hence, you gain reusability by means of four main object-oriented programming concepts. In order to clearly understand the object orientation model, let's take your " hand " as an example. The " hand " is a class.

By the way, there are several books written on Object-oriented design principles, design patterns, and best practices, but only a few of them provides what they claim. There are two things here, design principles and design patterns, one is basic and other is best practice built on that basis. One should first learn Object-oriented principles and then learn design patterns to see how those principles are used to solve day to day problems. Keeping this in mind, I have collected following a list of 5 books to learn Object-oriented basics and design patterns in Java. These books are ordered from beginners to advanced perspective. Top 5 books on Java Design Patterns Here is my list of good books to learn object-oriented and Java design patterns. Design patterns are tried and tested as a way to solve a problem in a context. These patterns are discovered while solving similar problems multiple times over the year and they address some of the most common tasks in object-oriented application development e. Following five books are the great resource to learn Java design patterns, which is equally applicable to any other object-oriented programming language. If you know any other book on design patterns, which is worth reading, then please share with us. Head First Design Patterns - Best Book to Learn Design Patterns This was the first book, I read on head first series and design pattern as well, prior to that, I have no idea when to use abstract class or interface or Why composition is better than Inheritance. I usually code each feature and requirement as they come and end of doing more changes, more testing and introducing more bugs in the first few years of software development. Thanks to my appetite for learning and reading, I discovered Head First design pattern and after reading its first chapter, I was thrilled. I would even suggest starting learning design pattern from this book. It not only explains concepts in a clear way, but also gives lots of diagrams, exercise, quizzes, and real-life examples to make you think and learn. The three chapters on the Decorator design pattern and Observer design pattern is also a great piece of work. In short, this is the best book on Java design pattern till date. Any list of must-read books on design patterns in Java is incomplete, without including this book. After almost 20 years, this book is still relevant in Object-oriented and Java design patterns. This book has a case study chapter on designing Document Editor and then explains various design patterns e. Creational patterns, Structural, and Behavioral patterns. A New Perspective on Object-Oriented Design Like multi-threading and concurrency, design patterns are also not easy to understand. What is the most difficult part is understanding enough to identify real-world scenarios, where you can apply these object-oriented design patterns. The main reason for that is the different style of writing and explaining stuff. Some programmer finds a particular author more readable than others, I guess the only exception is Joshua Bloch which really connects to most of Java developer. Though it also explains the same set of OOPS principles, UML and design patterns like decorator pattern , factory pattern or observer design pattern , the way it explains may be easier to understand for many beginners. Examples given this design pattern books are also good and nontrivial. Software Architecture Design Patterns in Java This is another good book on Object-oriented and Java design patterns, it not only covers basic Object-oriented principles like Class, Object, Inheritance, Polymorphism, Abstraction, and Encapsulation but also covers details which are quite practical but not obvious e. This book also provides extensive and comprehensive coverage of a whole lot of design patterns e. Creational patterns, Collection patterns, Structural patterns, Behavioral patterns, and Concurrency patterns. Another thing, which I like in this book is practice questions and UML diagrams, which not only helps to understand the topic well but also ensures that you practice application part, which is most important to learn any design pattern. Finally, they have a case study of designing software solution for a web hosting company, which gives you a real chance of identifying and applying design patterns in a real-world scenario. In short, Software Architecture Design Patterns in Java has almost everything you need to know about Java design patterns. Best Practices and Design Strategies The J2EE platform is the most popular way of using Java programming language, and since it mainly focuses on web and enterprise solution, it has its own set of problems and requirements. This book divides the design

pattern based upon their functional area e. Presentation tier design patterns, business tier design pattern etc. This is a must read the book for any Java J2EE developer, who is responsible for coding web application and enterprise application. Though modern day framework like Spring, ensures you follow these J2EE design pattern at the framework level e. Having said that, whether you use Spring MVC, Struts or any other web framework, knowledge of these J2EE patterns will help you to understand legacy code as well as to create a solution which is easier to maintain. I have read a couple of chapters of this book and I must say he was right, this is the most up-to-date book on the topic of Java EE Patterns and one of the must-read the book for Java EE developers. Apart from these 5 books, you can also look on Effective Java 2nd Edition by Joshua Bloch and Head First object-oriented analysis and design for getting some good idea about object-oriented design principles and how it has used in Java. Effective Java is the must-read for any Java programmer, as it explained a lot of practical design advice used in Java library itself.

Chapter 5 : Lesson: Object-Oriented Programming Concepts (The Javaâ„¢ Tutorials > Learning the Java L

This book is THE classic Gang of Four (GOF) design pattern book, which became source and motivation for many object-oriented design pattern books written and published so far. Any list of must-read books on design patterns in Java is incomplete, without including this book.

Features[edit] Object-oriented programming uses objects, but not all of the associated techniques and structures are supported directly in languages that claim to support OOP. The features listed below are, however, common among languages considered strongly class- and object-oriented or multi-paradigm with OOP support , with notable exceptions mentioned. Comparison of programming languages object-oriented programming and List of object-oriented programming terms Shared with non-OOP predecessor languages[edit] Variables that can store information formatted in a small number of built-in data types like integers and alphanumeric characters. This may include data structures like strings , lists , and hash tables that are either built-in or result from combining variables using memory pointers Procedures â€” also known as functions, methods, routines, or subroutines â€” that take input, generate output, and manipulate data. Modern languages include structured programming constructs like loops and conditionals. Modular programming support provides the ability to group procedures into files and modules for organizational purposes. Modules are namespaced so identifiers in one module will not be accidentally confused with a procedure or variable sharing the same name in another file or module. Objects and classes[edit] Languages that support object-oriented programming typically use inheritance for code reuse and extensibility in the form of either classes or prototypes. Those that use classes support two main concepts: Classes â€” the definitions for the data format and available procedures for a given type or class of object; may also contain data and procedures known as class methods themselves, i. For example, a graphics program may have objects such as "circle", "square", "menu". An online shopping system might have objects such as "shopping cart", "customer", and "product". It is essential to understand this; using classes to organize a bunch of unrelated methods together is not object orientation. Junade Ali, Mastering PHP Design Patterns [8] Each object is said to be an instance of a particular class for example, an object with its name field set to "Mary" might be an instance of class Employee. Procedures in object-oriented programming are known as methods ; variables are also known as fields , members, attributes, or properties. This leads to the following terms: Class variables â€” belong to the class as a whole; there is only one copy of each one Instance variables or attributes â€” data that belongs to individual objects; every object has its own copy of each one Member variables â€” refers to both the class and instance variables that are defined by a particular class Class methods â€” belong to the class as a whole and have access only to class variables and inputs from the procedure call Instance methods â€” belong to individual objects, and have access to instance variables for the specific object they are called on, inputs, and class variables Objects are accessed somewhat like variables with complex internal structure, and in many languages are effectively pointers , serving as actual references to a single instance of said object in memory within a heap or stack. They provide a layer of abstraction which can be used to separate internal from external code. External code can use an object by calling a specific instance method with a certain set of input parameters, read an instance variable, or write to an instance variable. Objects are created by calling a special type of method in the class known as a constructor. A program may create many instances of the same class as it runs, which operate independently. This is an easy way for the same procedures to be used on different sets of data. Object-oriented programming that uses classes is sometimes called class-based programming , while prototype-based programming does not typically use classes. As a result, a significantly different yet analogous terminology is used to define the concepts of object and instance. In some languages classes and objects can be composed using other concepts like traits and mixins. Class-based vs prototype-based[edit] In class-based languages the classes are defined beforehand and the objects are instantiated based on the classes. If two objects apple and orange are instantiated from the class Fruit, they are inherently fruits and it is guaranteed that you may handle them in the same way; e. In prototype-based languages the objects are the primary entities. No classes even exist. The prototype of an object is just another object to which the object is

linked. Every object has one prototype link and only one. New objects can be created based on already existing objects chosen as their prototype. You may call two different objects apple and orange a fruit, if the object fruit exists, and both apple and orange have fruit as their prototype. The attributes and methods of the prototype are delegated to all the objects of the equivalence class defined by this prototype. The attributes and methods owned individually by the object may not be shared by other objects of the same equivalence class; e. Only single inheritance can be implemented through the prototype. This feature is known as dynamic dispatch , and distinguishes an object from an abstract data type or module , which has a fixed static implementation of the operations for all instances. If the call variability relies on more than the single type of the object on which it is called i. A method call is also known as message passing. It is conceptualized as a message the name of the method and its input parameters being passed to the object for dispatch. Encapsulation[edit] Encapsulation is an object-oriented programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse. Data encapsulation led to the important OOP concept of data hiding. If a class does not allow calling code to access internal object data and permits access through methods only, this is a strong form of abstraction or information hiding known as encapsulation. Some languages Java, for example let classes enforce access restrictions explicitly, for example denoting internal data with the private keyword and designating methods intended for use by code outside the class with the public keyword. Methods may also be designed public, private, or intermediate levels such as protected which allows access from the same class and its subclasses, but not objects of a different class. In other languages like Python this is enforced only by convention for example, private methods may have names that start with an underscore. Encapsulation prevents external code from being concerned with the internal workings of an object. This facilitates code refactoring , for example allowing the author of the class to change how objects of that class represent their data internally without changing any external code as long as "public" method calls work the same way. It also encourages programmers to put all the code that is concerned with a certain set of data in the same class, which organizes it for easy comprehension by other programmers. Encapsulation is a technique that encourages decoupling. Composition, inheritance, and delegation[edit] Objects can contain other objects in their instance variables; this is known as object composition. Object composition is used to represent "has-a" relationships: Languages that support classes almost always support inheritance. This allows classes to be arranged in a hierarchy that represents "is-a-type-of" relationships. For example, class Employee might inherit from class Person. All the data and methods available to the parent class also appear in the child class with the same names. These will also be available in class Employee, which might add the variables "position" and "salary". This technique allows easy re-use of the same procedures and data definitions, in addition to potentially mirroring real-world relationships in an intuitive way. Rather than utilizing database tables and programming subroutines, the developer utilizes objects the user may be more familiar with: Multiple inheritance is allowed in some languages, though this can make resolving overrides complicated. Some languages have special support for mixins , though in any language with multiple inheritance, a mixin is simply a class that does not represent an is-a-type-of relationship. Mixins are typically used to add the same methods to multiple classes. Abstract classes cannot be instantiated into objects; they exist only for the purpose of inheritance into other "concrete" classes which can be instantiated. In Java, the final keyword can be used to prevent a class from being subclassed. The doctrine of composition over inheritance advocates implementing has-a relationships using composition instead of inheritance. For example, instead of inheriting from class Person, class Employee could give each Employee object an internal Person object, which it then has the opportunity to hide from external code even if class Person has many public attributes or methods. Some languages, like Go do not support inheritance at all. Delegation is another language feature that can be used as an alternative to inheritance. Polymorphism[edit] Subtyping - a form of polymorphism - is when calling code can be agnostic as to whether an object belongs to a parent class or one of its descendants. Meanwhile, the same operation name among objects in an inheritance hierarchy may behave differently. For example, objects of type Circle and Square are derived from a common class called Shape. The Draw function for each type of Shape implements what is necessary to draw itself while calling code can remain indifferent to the particular type of Shape is being drawn. This is another type of abstraction

which simplifies code external to the class hierarchy and enables strong separation of concerns. Open recursion[edit] In languages that support open recursion , object methods can call other methods on the same object including themselves , typically using a special variable or keyword called this or self. This variable is late-bound ; it allows a method defined in one class to invoke another method that is defined later, in some subclass thereof. History[edit] UML notation for a class. This Button class has variables for data, and functions. Through inheritance a subclass can be created as subset of the Button class. Objects are instances of a class. Terminology invoking "objects" and "oriented" in the modern sense of object-oriented programming made its first appearance at MIT in the late s and early s. In the environment of the artificial intelligence group, as early as , "object" could refer to identified items LISP atoms with properties attributes ; [10] [11] Alan Kay was later to cite a detailed understanding of LISP internals as a strong influence on his thinking in Alan Kay, [12] Another early MIT example was Sketchpad created by Ivan Sutherland in â€”61; in the glossary of the technical report based on his dissertation about Sketchpad, Sutherland defined notions of "object" and "instance" with the class concept covered by "master" or "definition" , albeit specialized to graphical interaction. For programming security purposes a detection process was implemented so that through reference counts a last resort garbage collector deleted unused objects in the random-access memory RAM. Simula launched in , and was promoted by Dahl and Nygaard throughout and , leading to increasing use of the programming language in Sweden, Germany and the Soviet Union. In , the language became widely available through the Burroughs B computers , and was later also implemented on the URAL computer. In , Dahl and Nygaard wrote a Simula compiler. They settled for a generalised process concept with record class properties, and a second layer of prefixes. Through prefixing a process could reference its predecessor and have additional properties. Simula thus introduced the class and subclass hierarchy, and the possibility of generating objects from these classes. The Simula 1 compiler and a new version of the programming language, Simula 67, was introduced to the wider world through the research paper "Class and Subclass Declarations" at a conference. By , the Association of Simula Users had members in 23 different countries. Early a Simula 67 compiler was released free of charge for the DecSystem mainframe family. The object-orientated Simula programming language was used mainly by researchers involved with physical modelling , such as models to study and improve the movement of ships and their content through cargo ports. Smaltalk included a programming environment and was dynamically typed , and at first was interpreted , not compiled. Smalltalk got noted for its application of object orientation at the language level and its graphical development environment. Smalltalk went through various versions and interest in the language grew. Experimentation with various extensions to Lisp such as LOOPS and Flavors introducing multiple inheritance and mixins eventually led to the Common Lisp Object System , which integrates functional programming and object-oriented programming and allows extension via a Meta-object protocol. In the s, there were a few attempts to design processor architectures that included hardware support for objects in memory but these were not successful. In , Goldberg edited the August issue of Byte Magazine , introducing Smalltalk and object-orientated programming to a wider audience.

Chapter 6 : âœ…{3 MB Latest} Balaguruswamy c++ PDF free download (Sixth Edition)

like \Object-oriented" are really a set of ideas and concepts that var- ious languages implement to varying degrees. Last term you toured around Computer Science (in FoCS) and used a.

This lesson will introduce you to objects, classes, inheritance, interfaces, and packages. Each discussion focuses on how these concepts relate to the real world, while simultaneously providing an introduction to the syntax of the Java programming language. What Is an Object? An object is a software bundle of related state and behavior. Software objects are often used to model the real-world objects that you find in everyday life. This lesson explains how state and behavior are represented within an object, introduces the concept of data encapsulation, and explains the benefits of designing your software in this manner. What Is a Class? A class is a blueprint or prototype from which objects are created. This section defines a class that models the state and behavior of a real-world object. It intentionally focuses on the basics, showing how even a simple class can cleanly model state and behavior. Inheritance provides a powerful and natural mechanism for organizing and structuring your software. This section explains how classes inherit state and behavior from their superclasses, and explains how to derive one class from another using the simple syntax provided by the Java programming language. What Is an Interface? An interface is a contract between a class and the outside world. When a class implements an interface, it promises to provide the behavior published by that interface. This section defines a simple interface and explains the necessary changes for any class that implements it. What Is a Package? A package is a namespace for organizing classes and interfaces in a logical manner. Placing your code into packages makes large software projects easier to manage. This section explains why this is useful, and introduces you to the Application Programming Interface API provided by the Java platform.

Chapter 7 : Java: Object-Oriented Programming Concepts [Integrated Course] | PACKT Books

Object-oriented programming System(OOPs) is a programming paradigm based on the concept of "objects" that contain data and methods. The primary purpose of object-oriented programming is to increase the flexibility and maintainability of programs.

As noted above, what makes the Object unique is that it combines data e. In most languages, Objects do not exist in source code: Instead you must create an Object piece-by-piece, filling-in the individual bits of data and code. It is important to remember that not only does the Object contain data, but it is also data itself. Once the program starts running, you can freely change the value of any piece of data inside the Object, and in most languages you can also change the elements data, code of the Object itself. Objects Typical Objects can be things that appear on the screen e. Class[edit] To use any value the number 7, the word Apple we need a data-type. Numbers have a data-type e. To define Objects and work with them in source-code, we need a data-type. Classes are the data-type for Objects. Each Class defines the specific set of data and the specific procedures that will make a particular Object. Classes as templates[edit] Because Objects are complex, with many different variables and procedures inside them, Classes are more complex than normal data-types. Most OOP languages give additional features to Classes to help us use them and create them. Not only does a Class define a data-type for an Object, but it usually defines default values for new Objects. This allows us to use Classes like a template, or biscuit-cutter: Classes and Objects You can use the same class as a template to make instantiate many objects Example: In OOP, we want to restrict some Procedures and Functions so that they can only act upon the data inside the Object they are attached to. These restricted versions have a special name: Every Method is a Procedure, with two special features: Attribute[edit] Data can be created anywhere, and used by any code at any time. In OOP, we want to restrict some data so that it can only be acted-upon by the Methods in the same Object. These restricted variables have a special name: Every Attribute is a Variable, with three special features: Attributes are further split into two types: All Fields and all Properties are Attributes, but with extra specialization. See the section on Encapsulation for the precise differences. Attributes In the example above we store the fuel and maxSpeed. Putting it together[edit] What is the difference between a class and an object? A class is a template which cannot be executed An object is an instance of a class which can be executed one class can be used to make many objects What are the main components of a class? What is the difference between a structure and a class Answer:

Chapter 8 : 5 Books to Learn Object Oriented Programming and Design Patterns - Best of lot

(views) Object-Oriented Programming with ANSI-C by Axel-Tobias Schreiner, In this book, we are going to use ANSI-C to discover how object-oriented programming is done, what its techniques are, why they help us solve bigger problems, and how we harness generality and program to catch mistakes earlier.

Chapter 9 : Python Class and Object-Oriented Concepts Explained with Examples

This book is a basic introduction to "Object Oriented Programming Using C#" for beginners who have never used C# before. After completing this book, you will understand.