DOWNLOAD PDF OBJECT LIFE CYCLES

Chapter 1 : Manage an Object's Lifecycle Using the Amazon S3 Console - Amazon Simple Storage Servic

In object-oriented programming (OOP), the object lifetime (or life cycle) of an object is the time between an object's creation and its destruction. Rules for object lifetime vary significantly between languages, in some cases between implementations of a given language, and lifetime of a particular object may vary from one run of the program to another.

The model affects outcome of If-Modified-Since IMS requests as well as various prefetching or validation algorithms that depend on object freshness. The model is configured using ObjLifeCycle type. Model components are described below. Object Creation Time Every Web object is assumed to be created some time in the past. In recent Polygraph distributions, the birthday is selected at random within a [time "zero", object life cycle length interval. Time zero is usually the beginning of year Thus, all objects are born way in the past unless cycle length is set to many years. The birthday corresponds to the beginning of the first life cycle, naturally. In old Polygraph versions, the birthday was determined using the user-specified birthday distribution and corresponded to the middle of the very first life cycle. Object Modification Times Polygraph assumes that Web objects or entities have a cyclic life style. That is, modifications happen with certain periodicity. For example, a daily news page may be modified every 24 hours, a personal home page may be stable for a month or so, and a page with old rock group lyrics might remain constant for years. Then a cycle period is defined as an average cycle length. We further observe that the period of a cycle is object specific. Modification pattern of a given object is usually stable and often independent from other objects. Clearly, for many objects, modifications do not happen at constant intervals. Variance field allows you to model variability of object modification times while keeping cycle period constant. The variability is expressed in percents of a cycle period. Zero percent means no variability; all modifications happen exactly at the middle of a cycle. Hundred percent variability means that, for a given cycle, an object may be modified at any time from the beginning until the end of a cycle. The picture below illustrates the object modification model. Note that we show several degrees of modification time variability, but the simulated variability is, of course, constant for a given object. All objects have known to Polygraph last modification times. However, real Web servers often do not include the Last-Modified: For a given object, Polygraph either always includes or always excludes the Last-Modified: Object Expiration Times Object expiration time is reported via the Expires: Since Polygraph knows future modification times of objects, it would be very easy to report precise expiration times, reducing the guess work on Web intermediaries. However, having this nice algorithm hard-coded into Polygraph would lead to unrealistic simulations. Indeed, real Web servers cannot predict future modification times. Hence, in most cases, servers lie about expiration time of objects. A server generates Expires: Usually, there is a way to tell a server to compute the Expires: The first formula expires all cached copies of a given object at the same absolute time. Polygraph server implements both formulas using the expires array and various time qualifiers lmt, now, nmt, etc. The portion of objects with unknown expiration time is calculated as a portion of objects not covered by the formulas in the expires array. For a given object, the presence of the Last-Modified: Furthermore, the presence and value of the Expires: This behavior mimics real world conditions. Note that the above assumes that generation of object modification times is enabled using the length field of ObjLifeCycle. Examples Here we give several typical applications of Object Life Cycle model. Most expiration times are easy and safe to predict. Life cycles differ a lot from object to object exponential distribution with 2 hour mean is used. For cachable objects, the time of last modification was set to about one year before the bake-off date. The expiration times was set to about one year after the bake-off date. Here is how that model can be configured using PGL.

DOWNLOAD PDF OBJECT LIFE CYCLES

Chapter 2 : What is the .NET object life cycle? - Stack Overflow

Life Cycle of an Object. 06/16/; 2 minutes to read Contributors. In this article. This topic describes the "life cycle" of an object, that is, how objects are referenced and tracked by the object manager.

Implementation System Analysis As in any other system development model, system analysis is the first phase of development in case of Object Modeling too. In this phase, the developer interacts with the user of the system to find out the user requirements and analyses the system to understand the functioning. Based on this system study, the analyst prepares a model of the desired system. This model is purely based on what the system is required to do. At this stage the implementation details are not taken care of. Only the model of the system is prepared based on the idea that the system is made up of a set of interacting objects. The important elements of the system are emphasized. System Design System Design is the next development stage where the overall architecture of the desired system is decided. The system is organized as a set of sub systems interacting with each other. While designing the system as a set of interacting subsystems, the analyst takes care of specifications as observed in system analysis as well as what is required out of the new system by the end user. As the basic philosophy of Object-Oriented method of system analysis is to perceive the system as a set of interacting objects, a bigger system may also be seen as a set of interacting smaller subsystems that in turn are composed of a set of interacting objects. While designing the system, the stress lies on the objects comprising the system and not on the processes being carried out in the system as in the case of traditional Waterfall Model where the processes form the important part of the system. Object Design In this phase, the details of the system analysis and system design are implemented. The Objects identified in the system design phase are designed. Here the implementation of these objects is decided as the data structures get defined and also the interrelationships between the objects are defined. Let us here deviate slightly from the design process and understand first a few important terms used in the Object-Oriented Modeling. As already discussed, Object Oriented Philosophy is very much similar to real world and hence is gaining popularity as the systems here are seen as a set of interacting objects as in the real world. To implement this concept, the process-based structural programming is not used; instead objects are created using data structures. Just as every programming language provides various data types and various variables of that type can be created, similarly, in case of objects certain data types are predefined. For example, we can define a data type called pen and then create and use several objects of this data type. This concept is known as creating a class. A class is a collection of similar objects. It is a template where certain basic characteristics of a set of objects are defined. The class defines the basic attributes and the operations of the objects of that type. Defining a class does not define any object, but it only creates a template. For objects to be actually created instances of the class are created as per the requirement of the case. Classes are built on the basis of abstraction, where a set of similar objects are observed and their common characteristics are listed. Of all these, the characteristics of concern to the system under observation are picked up and the class definition is made. The attributes of no concern to the system are left out. This is known as abstraction. The abstraction of an object varies according to its application. For instance, while defining a pen class for a stationery shop, the attributes of concern might be the pen color, ink color, pen type etc. Inheritance is another important concept in this regard. This concept is used to apply the idea of reusability of the objects. A new type of class can be defined using a similar existing class with a few new features. For instance, a class vehicle can be defined with the basic functionality of any vehicle and a new class called car can be derived out of it with a few modifications. This would save the developers time and effort as the classes already existing are reused without much change. Coming back to our development process, in the Object Designing phase of the Development process, the designer decides onto the classes in the system based on these concepts. The designer also decides on whether the classes need to be created from scratch or any existing classes can be used as it is or new classes can be inherited from them. Implementation During this phase, the class objects and the interrelationships of these classes are translated and actually coded using the programming language decided upon. The databases are made and the complete system is given a functional shape. The complete OO methodology revolves around the objects identified in

the system. When observed closely, every object exhibits some characteristics and behavior. The objects recognize and respond to certain events. For example, considering a Window on the screen as an object, the size of the window gets changed when resize button of the window is clicked. Here the clicking of the button is an event to which the window responds by changing its state from the old size to the new size. While developing systems based on this approach, the analyst makes use of certain models to analyze and depict these objects. The methodology supports and uses three basic Models: Object Model - This model describes the objects in a system and their interrelationships. This model observes all the objects as static and does not pay any attention to their dynamic nature. Dynamic Model - This model depicts the dynamic aspects of the system. It portrays the changes occurring in the states of various objects with the events that might occur in the system. Functional Model - This model basically describes the data transformations of the system. This describes the flow of data and the changes that occur to the data throughout the system. While the Object Model is most important of all as it describes the basic element of the system, the objects, all the three models together describe the complete functional system. As compared to the conventional system development techniques, OO modeling provides many benefits. Among other benefits, there are all the benefits of using the Object Orientation. Some of these are: Reusability - The classes once defined can easily be used by other applications. This is achieved by defining classes and putting them into a library of classes where all the classes are maintained for future use. Whenever a new class is needed the programmer looks into the library of classes and if it is available, it can be picked up directly from there. Inheritance - The concept of inheritance helps the programmer use the existing code in another way, where making small additions to the existing classes can quickly create new classes. Programmer has to spend less time and effort and can concentrate on other aspects of the system due to the reusability feature of the methodology. Data Hiding - Encapsulation is a technique that allows the programmer to hide the internal functioning of the objects from the users of the objects. Encapsulation separates the internal functioning of the object from the external functioning thus providing the user flexibility to change the external behaviour of the object making the programmer code safe against the changes made by the user. The systems designed using this approach are closer to the real world as the real world functioning of the system is directly mapped into the system designed using this approach. Because of this, it is easier to produce and understand designs. The objects in the system are immune to requirement changes. Therefore, allows changes more easily. Object Oriented Methodology designs encourage more re-use. New applications can use the existing modules, thereby reduces the development cost and cycle time. Object Oriented Methodology approach is more natural. It provides nice structures for thinking and abstracting and leads to modular design.

DOWNLOAD PDF OBJECT LIFE CYCLES

Chapter 3 : Spring Bean Life Cycle Explained - HowToDoInJava

This course will introduce students to the basics of the Structured Query Language (SQL) as well as basic database design for storing data as part of a multi-step data gathering, analysis, and processing effort.

Lifecycle configuration on MFA-enabled buckets is not supported. You are not charged for storage time associated with an object that has expired. AWS Certification Exam Practice Questions Questions are collected from Internet and the answers are marked as per my knowledge and understanding which might differ with yours. AWS services are updated everyday and both the answers and questions might be outdated soon, so research accordingly. AWS exam questions are not updated to keep up the pace with AWS updates, so even if the underlying feature has changed the question might not be updated Open to further feedback, discussion and correction. If an object is stored in the Standard S3 storage class and you want to move it to Glacier, what must you do in order to properly migrate it? Change the storage class directly on the object. Delete the object and re-upload it, selecting Glacier as the storage class. None of the above. Create a lifecycle policy that will migrate it after a minimum of 30 days. Any object uploaded to S3 must first be placed into either the Standard, Reduced Redundancy, or Infrequent Access storage class. Once in S3 the only way to move the object to glacier is through a lifecycle policy A company wants to store their documents in AWS. Initially, these documents will be used frequently, and after a duration of 6 months, they would not be needed anymore. How would you architect this requirement? Store the files in Amazon S3 and create a Lifecycle Policy to remove the files after 6 months. Store the files in Amazon Glacier and create a Lifecycle Policy to remove the files after 6 months. Your firm has uploaded a large amount of aerial image data to S3. In the past, in your on-premises environment, you used a dedicated group of servers to oaten process this data and used Rabbit MQ, an open source messaging system, to get job information to the servers. Once processed the data would go to tape and be shipped offsite. Your manager told you to stay with the current design, and leverage AWS archival storage and messaging services to minimize cost. Once data is processed, change the storage class of the S3 objects to Glacier Glacier suitable for Tape backup Use SNS to pass job messages use Cloud Watch alarms to terminate spot worker instances when they become idle. Once data is processed, change the storage class of the S3 object to Glacier. You have a proprietary data store on-premises that must be backed up daily by dumping the data store contents to a single compressed 50GB file and sending the file to AWS. Your SLAs state that any dump file backed up within the past 7 days can be retrieved within 2 hours. Your compliance department has stated that all data must be held indefinitely. The time required to restore the data store from a backup is approximately 1 hour. Your on-premise network connection is capable of sustaining 1gbps to AWS. Which backup methods to AWS would be most cost-effective while still meeting all of your requirements? Send the daily backup files to Glacier immediately after being generated will not meet the RTO Transfer the daily backup files to an EBS volume in AWS and take daily snapshots of the volume Not cost effective Transfer the daily backup files to S3 and use appropriate bucket lifecycle policies to send to Glacier Store in S3 for seven days and then archive to Glacier Host the backup files on a Storage Gateway with Gateway-Cached Volumes and take daily snapshots Not Cost effective as local storage as well as S3 storage.

DOWNLOAD PDF OBJECT LIFE CYCLES

Chapter 4 : Managing Object Lifecycles | Cloud Storage | Google Cloud

As you work with objects in Java, understanding how objects are born, live their lives, and die is important. This topic is called the life cycle of an object, and it goes something like this: 1. Before an object can be created from a class, the class must be loaded. To do that, the Java runtime.

Generics Before a Java object can be created the class byte code must be loaded from the file system with. This process of locating the byte code for a given class name and converting that code into a Java class instance is known as class loading. There is one class created for each type of Java class. All objects in Java programs are created on heap memory. An object is created based on its class. You can consider a class as a blueprint, template, or a description how to create an object. When an object is created, memory is allocated to hold the object properties. An object reference pointing to that memory location is also created. To use the object in the future, that object reference has to be stored as a local variable or as an object member variable. If there are no more reference to the object, the object can not be used any more and becomes garbage. After a while the heap memory will be full of unused objects. The JVM collects those garbage objects and frees the memory they allocated, so the memory can be reused again when a new object is created. See below a simple example: Object references can be passed in to methods and can be returned from methods. Depending on which package the class belongs to, the package name will be translated to a directory path. The JVM stores the code in memory, allocates memory for the static variables, and executes any static initialize block. Memory is not allocated for the object member variables at this point, memory will be allocated for them when an instance of the class, an object, is created. There is no limit on how many objects from the same class can be created. Code and static variables are stored only once, no matter how many objects are created. Memory is allocated for the object member variables when the object is created. Creating object by cloning an object[edit] Cloning is not automatically available to classes. There is some help though, as all Java objects inherit the protected Object clone method. You may ask why we need this clone method. An example would be note that accessing the private memberVar variable of obj is legal as this is in the same class: See below a factory method that will return a new object using cloning. First the Customer class needs to implement the Cloneable Interface. Override and make the clone method public , as that is protected in the Object class. Override the clone method in all the subclasses of Customer. Another use of cloning could be to take a snapshot of an object that can change in time. Now we are facing a problem, it is not enough to clone the Customer object, we also need to clone the referenced objects. References to unchangeable objects such as a String can be used in the cloned object without worry. Re-creating an object received from a remote source[edit] When an object is sent through a network, the object needs to be re-created at the receiving host. Object Serialization The term Object Serialization refers to the act of converting the object to a byte stream. The byte stream can be stored on the file system or can be sent through a network. At a later time the object can be re-created from that stream of bytes. The only requirement is that the same class has to be available both times, when the object is serialized and also when the object is re-created. If that happens on different servers, then the same class must be available on both servers. This is a maintenance problem for those applications where java serialization is used to make objects persistent or to sent the object through the network. When a class is modified, there could be a problem re-creating those objects that were serialized using an earlier version of the class. Java has built-in support for serialization, using the Serializable interface; however, a class must first implement the Serializable interface. By default, a class will have all of its fields serialized when converted into a data stream with transient fields being skipped. If additional handling is required beyond the default of writing all fields, you need to provide an implementation for the following three methods: You can still allow the class to be loaded by defining the serialization version id: To accomplish this, the finalize method is used. Also, we can not rely on when the finalize method will be called. If the java application exits before the object is garbage-collected, the finalize method may never be called. If it fails for some reason, it is ignored. If the application creates objects faster than the garbage-collector can claim back memory, the program can run out of memory. The finalize method is required only if there are resources

beyond the direct control of the Java Virtual Machine that needs to be cleaned up. In particular, there is no need to explicitly close an OutputStream, since the OutputStream will close itself when it gets finalized. Instead, the finalize method is used to release either native or remote resources controlled by the class. Class loading[edit] One of the main concerns of a developer writing hot re-deployable applications is to understand how class loading works. Within the internals of the class loading mechanism lies the answer to questions like: How can I use two different versions of the same utility library, simultaneously, within the same instance of the application server? What version of an utility class I am currently using? Why do I need to mess with all this class loading stuff anyway?

Chapter 5 : What is. Life cycle of an object

The Object Life Cycle. Creating an object: You use the new keyword to instantiate the new object. A block of memory is allocated. This block of memory is big enough.

History[edit] In the early days of object-oriented technology before the mids, there were many different competing methodologies for software development and object-oriented modeling , often tied to specific Computer Aided Software Engineering CASE tool vendors. No standard notations, consistent terms and process guides were the major concerns at the time, which degraded communication efficiency and lengthened learning curves. Later, together with Philippe Kruchten and Walker Royce eldest son of Winston Royce , they have led a successful mission to merge their own methodologies, OMT , OOSE and Booch method , with various insights and experiences from other industry leaders into the Rational Unified Process RUP , a comprehensive iterative and incremental process guide and framework for learning industry best practices of software development and project management. The specific problem is: January Learn how and when to remove this template message The software life cycle is typically divided up into stages going from abstract descriptions of the problem to designs then to code and testing and finally to deployment. The earliest stages of this process are analysis and design. The analysis phase is also often called "requirements acquisition". In some approaches to software developmentâ€”known collectively as waterfall modelsâ€”the boundaries between each stage are meant to be fairly rigid and sequential. The term "waterfall" was coined for such methodologies to signify that progress went sequentially in one direction only, i. The alternative to waterfall models are iterative models. This distinction was popularized by Barry Boehm in a very influential paper on his Spiral Model for iterative software development. With iterative models it is possible to do work in various stages of the model in parallel. So for example it is possibleâ€”and not seen as a source of errorâ€”to work on analysis, design, and even code all on the same day and to have issues from one stage impact issues from another. As a result, in object-oriented processes "analysis and design" are often considered at the same time. The object-oriented paradigm emphasizes modularity and re-usability. The goal of an object-oriented approach is to satisfy the "open closed principle". A module is open if it supports extension. If the module provides standardized ways to add new behaviors or describe new states. In the object-oriented paradigm this is often accomplished by creating a new subclass of an existing class. A module is closed if it has a well defined stable interface that all other modules must use and that limits the interaction and potential errors that can be introduced into one module by changes in another. In the object-oriented paradigm this is accomplished by defining methods that invoke services on objects. Methods can be either public or private, i. This reduces a source of many common errors in computer programming. The distinction between analysis and design is often described as "what vs. In analysis developers work with users and domain experts to define what the system is supposed to do. Implementation details are supposed to be mostly or totally depending on the particular method ignored at this phase. The goal of the analysis phase is to create a functional model of the system regardless of constraints such as appropriate technology. In object-oriented analysis this is typically done via use cases and abstract definitions of the most important objects. The subsequent design phase refines the analysis model and makes the needed technology and other implementation choices. In object-oriented design the emphasis is on describing the various objects, their data, behavior, and interactions. The design model should have all the details required so that programmers can implement the design in code. The main difference between object-oriented analysis and other forms of analysis is that by the object-oriented approach we organize requirements around objects, which integrate both behaviors processes and states data modeled after real world objects that the system interacts with. In other or traditional analysis methodologies, the two aspects: For example, data may be modeled by ER diagrams , and behaviors by flow charts or structure charts. The primary tasks in object-oriented analysis OOA are: Find the objects Describe how the objects interact Define the behavior of the objects Define the internals of the objects Common models used in OOA are use cases and object models. Use cases describe scenarios for standard domain functions that the system must accomplish. Object models describe the names, class relations e. Circle is a subclass of Shape , operations, and

properties of the main objects. User-interface mockups or prototypes can also be created to help understanding. Object-oriented design During object-oriented design OOD , a developer applies implementation constraints to the conceptual model produced in object-oriented analysis. Such constraints could include the hardware and software platforms, the performance requirements, persistent storage and transaction, usability of the system, and limitations imposed by budgets and time. Concepts in the analysis model which is technology independent, are mapped onto implementing classes and interfaces resulting in a model of the solution domain, i. Object-oriented modeling[edit] Object-oriented modeling OOM is a common approach to modeling applications, systems, and business domains by using the object-oriented paradigm throughout the entire development life cycles. Object-oriented modeling typically divides into two aspects of work: Efficient and effective communication Users typically have difficulties in understanding comprehensive documents and programming language codes well. Visual model diagrams can be more understandable and can allow users and stakeholders to give developers feedback on the appropriate requirements and structure of the system. A key goal of the object-oriented approach is to decrease the "semantic gap" between the system and the real world, and to have the system be constructed using terminology that is almost the same as the stakeholders use in everyday business. Object-oriented modeling is an essential tool to facilitate this. Useful and stable abstraction Modeling helps coding. A goal of most modern software methodologies is to first address "what" questions and then address "how" questions, i. Object-oriented modeling enables this by producing abstract and accessible descriptions of both system requirements and designs, i.

Chapter 6 : Object Life-cycle Overview

In general the "life cycle" of an object (for example in a Inversion Of Control context) is its duration and its availability into the application.

Expand Object lifetime is the time when a block of memory is allocated to this object during some process of execution and that block of memory is released when the process ends. Once the object is allocated with memory, it is necessary to release that memory so that it is used for further processing, otherwise it would result in memory leaks. We have a class in. Net that releases memory automatically for us when the object is no longer used. We will try to understand the entire scenario thoroughly of how objects are created and allocated memory and then deallocated when the object is out of scope. The class is a Blueprint that describes how an instance of this type will look and feel in memory. This instance is the object of that class type. A block of memory is allocated when the new keyword is used to instantiate the new object and constructor is called. This block of memory is big enough to hold the object. When we declare a class variable it is allocated on the stack and the time it hits a new keyword and then it is allocated on the heap. In other words, when an object of a class is created it is allocated on the heap with the C new keyword operator. However a new keyword returns a reference to the object on the heap, not the actual object itself. This reference variable is stored on the stack for further use in applications. When the new operator is used to create an object, memory is taken from the managed heap for this object and the managed heap is more than just a random chunk of memory accessed by the CLR. When the object is no longer used then it is de-allocated from the memory so that this memory can be reused. The key pillar of the. NET Framework is the automatic garbage collection that manages memory for all. When an object is instantiated the garbage collector will destroy the object when it is no longer needed. There is no explicit memory deal location since the garbage collector monitors unused objects and does a collection to free up memory that is an automatic process. The Garbage Collector removes objects from the heap when they are unreachable by any part of your code base. Net garbage collector will compact empty blocks of memory for the purpose of optimization. The heap is categorized into three generations so it can handle long-lived and short-lived objects. Garbage collection primarily occurs with the reclamation of short-lived objects that typically occupy only a small part of the heap. Generations There are the following three generations of objects on the heap: Newly created objects are at Generation 0. These objects on Generation 0 are collected frequently to ensure that short-lived objects are quickly collected and the memory is released. Objects that survive Generation 0, the collections are promoted to Generation 1. Most objects are reclaimed for garbage collection in Generation 0 and do not survive to the next generation. Objects that are collected less frequently than Generation 0 and contains longer-lived objects that were promoted from Generation 0. Objects that survive Generation 1, collection are promoted to Generation 2. Objects promoted from Generation 1 that are the longest-lived objects and collected infrequently. The overall strategy of the garbage collector is to collect and move longer-lived objects less frequently. The garbage collector cleans up managed resources automatically since managed code is directly targeted by the CLR. But when the object uses unmanaged resources like database connections or file manipulation, that needs to be released manually and this can be done by a finalize method. This is known as a finalization process. If we are implementing Finalize , we do not have control since when this method should be called the garbage collector takes care of this on its own. During the first collection pass, the object is flagged for finalization. There is another method, Dispose , that releases managed and unmanaged resources explicitly. This method is the single method in an IDisposable interface and can be used to release unmanaged resources manually. I have written another article on the IDisposable pattern in which we would have a clear picture of the difference between Finalize and Dispose.

Chapter 7 : Object Lifetime in .NET Framework

A framework object's "life cycle" spans the time from when an object is created to when it is deleted. An object's reference count controls when it will be deleted. Most framework objects are created by a driver's call to the object's creation method. For example, each framework driver must call.

Overview[edit] While the basic idea of object lifetime is simple â€” an object is created, used, then destroyed â€” details vary substantially between languages, and within implementations of a given language, and is intimately tied to how memory management is implemented. Further, many fine distinctions are drawn between the steps, and between language-level concepts and implementation-level concepts. Terminology is relatively standard, but which steps correspond to a given term varies significantly between languages. This varies by language, and within language varies with the memory allocation of an object; object lifetime may be distinct from variable lifetime. Objects with static memory allocation , notably objects stored in static variables , and classes modules if classes or modules are themselves objects, and statically allocated , have a subtle non-determinism in many languages: In garbage-collected languages, objects are generally dynamically allocated on the heap even if they are initially bound to an automatic variable, unlike automatic variables with primitive values, which are typically automatically allocated on the stack or in a register. This allows the object to be returned from a function "escape" without being destroyed. However, in some cases a compiler optimization is possible, namely performing escape analysis and proving that escape is not possible, and thus the object can be allocated on the stack; this is significant in Java. A complex case is the use of an object pool , where objects may be created ahead of time or reused, and thus apparent creation and destruction may not correspond to actual creation and destruction of an object, only re initialization for creation and finalization for destruction. In this case both creation and destruction may be nondeterministic. Steps[edit] Object creation can be broken down into two operations: These are implementation-level concepts, roughly analogous to the distinction between declaration and initialization or definition of a variable, though these later are language-level distinctions. For an object that is tied to a variable, declaration may be compiled to memory allocation reserving space for the object , and definition to initialization assigning values , but declarations may also be for compiler use only such as name resolution , not directly corresponding to compiled code. Analogously, object destruction can be broken down into two operations, in the opposite order: These do not have analogous language-level concepts for variables: Together these yield four implementation-level steps: Initialization is very commonly programmer-specified in class-based languages , while in strict prototype-based languages initialization is automatically done by copying. Allocation is more rarely specified, and deallocation generally cannot be specified. Status during creation and destruction[edit] An important subtlety is the status of an object during creation or destruction, and handling cases where errors occur or exceptions are raised, such as if creation or destruction fail. Thus during initialization and finalization an object is alive, but may not be in a consistent state â€” ensuring class invariants is a key part of initialization â€” and the period from when initialization completes to when finalization starts is when the object is both alive and expected to be in a consistent state. If creation or destruction fail, error reporting often by raising an exception can be complicated: The opposite issue â€” incoming exceptions, not outgoing exceptions â€” is whether creation or destruction should behave differently if they occur during exception handling, when different behavior may be desired. Another subtlety is when creation and destruction happen for static variables , whose lifespan coincides with the run time of the program â€” do creation and destruction happen during regular program execution, or in special phases before and after regular execution â€” and how objects are destroyed at program termination, when the program may not be in a usual or consistent state. This is particularly an issue for garbage-collected languages, as they may have a lot of garbage at program termination. Class-based programming[edit] In class-based programming, object creation is also known as instantiation creating an instance of a class , and creation and destruction can be controlled via methods known as a constructor and destructor , or an initializer and finalizer. Creation and destruction are thus also known as construction and destruction, and when these methods are called an object is said to be constructed or

destructed not "destroyed" â€” respectively, initialized or finalized when those methods are called. A key distinction is that constructors are class methods, as there is no object class instance available until the object is created, but the other methods destructors, initializers, and finalizers are instance methods, as an object has been created. Further, constructors and initializers may take arguments, while destructors and finalizers generally do not, as they are usually called implicitly. The steps during finalization vary significantly depending on memory management: Thus in automatic reference counting, programmer-specified finalizers are often short or absent, but significant work may still be done, while in tracing garbage collectors finalization is often unnecessary. Resource management[edit] In languages where objects have deterministic lifetimes, object lifetime can also be used for resource management , notably via the Resource Acquisition Is Initialization RAII idiom: In languages where objects have non-deterministic lifetimes, notably due to garbage collection, resources are managed in other ways, notably the dispose pattern: Using object lifetime for resource management ties memory management to resource management, and thus is not generally used in garbage-collected languages, as it would either constrain the garbage collector requiring immediate finalization or result in possibly long-lasting resource leaks , due to finalization being deferred. Object creation[edit] In typical case, the process is as follows: When the object in question is not derived from a class, but from a prototype instead, the size of an object is usually that of the internal data structure a hash for instance that holds its slots. For example, in multi-inheritance , which initializing code should be called first is a difficult question to answer. However, superclass constructors should be called before subclass constructors. It is a complex problem to create each object as an element of an array. Handling exceptions in the midst of creation of an object is particularly problematic because usually the implementation of throwing exceptions relies on valid object states. For instance, there is no way to allocate a new space for an exception object when the allocation of an object failed before that due to a lack of free space on the memory. Due to this, implementations of OO languages should provide mechanisms to allow raising exceptions even when there is short supply of resources, and programmers or the type system should ensure that their code is exception-safe. Propagating an exception is more likely to free resources than to allocate them. But in object oriented programming, object construction may fail, because constructing an object should establish the class invariants , which are often not valid for every combination of constructor arguments. Thus, constructors can raise exceptions. The abstract factory pattern is a way to decouple a particular implementation of an object from code for the creation of such an object. Creation methods[edit] The way to create objects varies across languages. In some class-based languages, a special method known as a constructor , is responsible for validating the state of an object. Just like ordinary methods, constructors can be overloaded in order to make it so that an object can be created with different attributes specified. Also, the constructor is the only place to set the state of immutable objects [Wrong clarification needed]. Other programming languages, such as Objective-C , have class methods, which can include constructor-type methods, but are not restricted to merely instantiating objects. This can be problematic if the programmer wants to provide two constructors with the same argument types, e. However, if there is sufficient memory or a program has a short run time, object destruction may not occur, memory simply being deallocated at process termination. In some cases object destruction simply consists of deallocating the memory, particularly in garbage-collected languages, or if the "object" is actually a plain old data structure. In other cases some work is performed prior to deallocation, particularly destroying member objects in manual memory management , or deleting references from the object to other objects to decrement reference counts in reference counting. This may be automatic, or a special destruction method may be called on the object. In garbage collecting languages, objects may be destroyed when they can no longer be reached by the running code. In class-based GCed languages, the analog of destructors are finalizers , which are called before an object is garbage-collected. Example of such languages include Java , Python , and Ruby. Destroying an object will cause any references to the object to become invalid, and in manual memory management any existing references become dangling references. In garbage collection both tracing garbage collection and reference counting , objects are only destroyed when there are no references to them, but finalization may create new references to the object, and to prevent dangling references, object resurrection occurs so the references remain valid. Integer ; constructor

DOWNLOAD PDF OBJECT LIFE CYCLES

CreateCopy APoint:

Chapter 8 : Object Lifecycle - Wikibooks, open books for an open world

Course 4 of 4 in the Specialization Web Applications for Everybody In this course, we'll look at the JavaScript language, and how it supports the Object-Oriented pattern, with a focus on the unique aspect of how JavaScript approaches OO. We'll explore a brief introduction to the jQuery library.

Chapter 9 : Object lifetime - Wikipedia

Object life cycle model is responsible for simulating object modifications, expirations, and similar events in Web object's ``life''. The model affects outcome of If-Modified-Since (IMS) requests as well as various prefetching or validation algorithms that depend on object freshness.