

DIAGRAMAS DE REFERENCIA Y DIAGRAMAS GENERADOS

ID	Diagramas de Referencia		Diagramas Generados	
	Código en PlantUML	Diagrama en PlantUML	Código en PlantUML	Diagrama en PlantUML
HU-CL01	<pre>@startuml class Member { memberId : Int name : String email : String } class Profile { profileId : Int info : String tags : List<String> } Member "1" -- "1" Profile @enduml</pre>	<pre> classDiagram class Member { memberId : Int name : String email : String } class Profile { profileId : Int info : String tags : List<String> } Member "1" -- "1" Profile </pre>	<pre> @startuml class Member { memberId : Int + updateProfile() } class Profile { } class ProfileField { - name: String - value: String } class Page { } Member "1" *-- "1" Profile Profile "1" *-- "*" ProfileField Profile --> Page @enduml </pre>	<pre> classDiagram class Member { + updateProfile() } class Profile { } class ProfileField { - name: String - value: String } class Page { } Member "1" *-- "1" Profile Profile "1" *-- "*" ProfileField Profile --> Page </pre>

HU-CL02	<pre>@startuml class Practitioner { practitionerId : Int name : String } class Profile { profileId : Int experience : String } Practitioner "1" -- "1" Profile @enduml</pre>	<pre> classDiagram class Practitioner { practitionerId : Int name : String } class Profile { profileId : Int experience : String } Practitioner "1" .. "1" Profile </pre>	<pre>@startuml class Member { } class Practitioner { + addExperience(experience: Experience): void } class Profile { } class Experience { - details: String } Practitioner --> Member Member "1" *-- "1" Profile Profile "1" *-- "*" Experience @enduml</pre>	<pre> classDiagram class Practitioner { <<addExperience(experience: Experience): void>> } class Member class Profile class Experience { <<details: String>> } Practitioner --> Member Member "1" *-- "1" Profile Profile "1" *-- "*" Experience </pre>
HU-CL03	<pre>@startuml class Trainer{ trainerId : Int name : String } class Profile { profileId : Int additional_info : String } Practitioner "1" -- "1" Profile @enduml</pre>	<pre> classDiagram class Trainer { trainerId : Int name : String } class Profile { profileId : Int additional_info : String } Trainer "1" .. "1" Profile </pre>	<pre>@startuml class Member { } class Trainer { + includeDetails(): void } class Profile { - details: String } Trainer --> Member Member "1" *-- "1" Profile @enduml</pre>	<pre> classDiagram class Trainer { <<includeDetails(): void>> } class Member class Profile { <<details: String>> } Trainer --> Member </pre>

HU-CL04	<pre>@startuml class Member { memberId : Int name : String email : String } class Profile { profileId : Int summary : String isPrivate : Boolean } class ProfileDirectory Member "1" -- "1" Profile : owns > ProfileDirectory ..> Profile : lists > @enduml</pre>	<pre> classDiagram class Member { memberId : Int name : String email : String } class Profile { profileId : Int summary : String isPrivate : Boolean } class ProfileDirectory Member "1" -- "1" Profile : owns ProfileDirectory ..> Profile : lists @enduml </pre> <p>The diagram shows three classes: Member, Profile, and ProfileDirectory. Member has attributes memberId, name, and email. Profile has attributes profileId, summary, and isPrivate. ProfileDirectory is associated with Profile via a dashed line labeled 'lists'. Member is associated with Profile via a solid line labeled 'owns'.</p>	<pre>@startuml class Member { + viewProfile(profile: Profile) + connectWith(member: Member) } class Profile {} class Connection {} Member "1" -- "1" Profile Member ..> Profile (Member, Member) .. Connection @enduml</pre> <p>The diagram shows Member, Profile, and Connection classes. Member has methods viewProfile and connectWith. Profile and Connection are empty classes. A self-loop association '1' is shown on Member, and a directed association '1'..> 'Profile' is shown from Member to Profile. A bidirectional association '(Member, Member) .. Connection' is shown between Member and Connection.</p>	<pre> classDiagram class Member { + viewProfile(profile: Profile) + connectWith(member: Member) } class Profile {} class Connection {} Member "1" -- "1" Profile Member ..> Profile (Member, Member) .. Connection @enduml </pre> <p>The diagram shows Member, Profile, and Connection classes. Member has methods viewProfile and connectWith. Profile and Connection are empty classes. A self-loop association '1' is shown on Member, and a directed association '1'..> 'Profile' is shown from Member to Profile. A bidirectional association '(Member, Member) .. Connection' is shown between Member and Connection.</p>
HU-CL05	<pre>@startuml class Member { memberId : Int name : String email : String } class Profile { profileId : Int name : String summary : String isPrivate : Boolean } class SearchCriteria { name : String tag : String location : String } class ProfileSearchService Member "1" -- "1" Profile : owns</pre>	<pre> classDiagram class Member { memberId : Int name : String email : String } class Profile { profileId : Int name : String summary : String isPrivate : Boolean } class SearchCriteria { name : String tag : String location : String } class ProfileSearchService Member "1" -- "1" Profile : owns ProfileSearchService ..> Profile : returns matches ProfileSearchService ..> SearchCriteria : uses @enduml </pre> <p>The diagram shows five classes: Member, Profile, SearchCriteria, ProfileSearchService, and a unnamed class represented by a rectangle with a dashed border. Member has attributes memberId, name, and email. Profile has attributes profileId, name, summary, and isPrivate. SearchCriteria has attributes name, tag, and location. ProfileSearchService is associated with Profile via a dashed line labeled 'returns matches' and with SearchCriteria via a dashed line labeled 'uses'.</p>	<pre>@startuml class Member { + searchForProfiles() + requestConnection(member: Member) } class Profile {} class ProfileField { - name: String - value: String } class Connection {} Member "1" -- "1" Profile Profile "1" *-- "*" ProfileField Member "2" -- "1" Connection @enduml</pre> <p>The diagram shows Member, Profile, ProfileField, and Connection classes. Member has methods searchForProfiles and requestConnection. Profile and Connection are empty classes. Profile has a multiplicity '*' at one end of its association with ProfileField. Member has a multiplicity '2' at one end of its association with Connection.</p>	<pre> classDiagram class Member { + searchForProfiles() + requestConnection(member: Member) } class Profile {} class ProfileField { # name: String # value: String } class Connection {} Member "1" -- "1" Profile Profile "1" *-- "*" ProfileField Member "2" -- "1" Connection @enduml </pre> <p>The diagram shows Member, Profile, ProfileField, and Connection classes. Member has methods searchForProfiles and requestConnection. Profile and Connection are empty classes. Profile has a multiplicity '*' at one end of its association with ProfileField. Member has a multiplicity '2' at one end of its association with Connection.</p>

	<pre>> ProfileSearchService ..> SearchCriteria : uses > ProfileSearchService ..> Profile : returns matches @enduml</pre>		
HU-CL06	<pre>@startuml class Member { memberId : Int name : String email : String } class Profile { profileId : Int info: String isPrivate : Boolean } Member "1" -- "1" Profile @enduml</pre>	<pre> classDiagram class Member { memberId : Int name : String email : String } class Profile { profileId : Int info: String isPrivate : Boolean } Member "1" -- "1" Profile </pre> <p>The diagram shows two classes: Member and Profile. Member has attributes memberId (Int), name (String), and email (String). Profile has attributes profileId (Int), info (String), and isPrivate (Boolean). A 1-to-1 relationship connects Member and Profile.</p>	<pre> @startuml class Member { + markProfileAsPrivate(): void } class Profile { } class PrivacySetting { - status: String } Member "1" *-- "1" Profile Profile "1" *-- "1" PrivacySetting @enduml </pre> <p>The code defines three classes: Member, Profile, and PrivacySetting. Member has a method markProfileAsPrivate(). Profile and PrivacySetting are empty classes. A 1-to-1 relationship connects Member and Profile, and another 1-to-1 relationship connects Profile and PrivacySetting.</p> <pre> classDiagram class Member { + markProfileAsPrivate(): void } class Profile { } class PrivacySetting { - status: String } Member "1" *-- "1" Profile Profile "1" *-- "1" PrivacySetting </pre> <p>The diagram shows three classes: Member, Profile, and PrivacySetting. Member has a method markProfileAsPrivate(). Profile and PrivacySetting are empty classes. A 1-to-1 relationship connects Member and Profile, and another 1-to-1 relationship connects Profile and PrivacySetting.</p>

HU-CL07	<pre>@startuml class Member { memberId : Int name : String email : String } class Profile { profileId : Int info : String emailPrivate : Boolean } Member "1" -- "1" Profile @enduml</pre>	<pre> classDiagram class Member { memberId : Int name : String email : String } class Profile { profileId : Int info : String emailPrivate : Boolean } Member "1" -- "1" Profile </pre> <p>The diagram shows two classes: Member and Profile. Member has attributes memberId (Int), name (String), and email (String). Profile has attributes profileId (Int), info (String), and emailPrivate (Boolean). A 1-to-1 relationship connects Member and Profile.</p>	<pre>@startuml class Member { + setEmailPrivacy(setting: PrivacySetting) } class Profile { } class EmailAddress { - address: String } class PrivacySetting { - level: String } Member "1" -- "1" Profile Profile "1" -- "1" PrivacySetting Profile "1" *-- "1" EmailAddress EmailAddress "1" -- "1" PrivacySetting @enduml</pre>	<pre> classDiagram class Member { + setEmailPrivacy(setting: PrivacySetting) } class Profile { } class EmailAddress { - address: String } class PrivacySetting { - level: String } Member "1" -- "1" Profile Profile "1" -- "1" PrivacySetting Profile "1" *-- "1" EmailAddress EmailAddress "1" -- "1" PrivacySetting </pre> <p>This diagram includes the original classes and their associations. It also adds the setEmailPrivacy method to the Member class and the address attribute to the EmailAddress class.</p>
HU-CL08	<pre>@startuml class Trainer { trainerId : Int name : String email : String } class Profile { profileId : Int info : String } class Course { courseId : Int title : String detailsURL : String } Trainer "1" -- "1" Profile Trainer "1" -- "0..*" Course @enduml</pre>	<pre> classDiagram class Trainer { trainerId : Int name : String email : String } class Profile { profileId : Int info : String } class Course { courseId : Int title : String detailsURL : String } Trainer "1" -- "1" Profile Trainer "1" -- "0..*" Course </pre> <p>The diagram shows three classes: Trainer, Profile, and Course. Trainer has attributes trainerId (Int), name (String), and email (String). Profile has attributes profileId (Int) and info (String). Course has attributes courseId (Int), title (String), and detailsURL (String). A 1-to-1 relationship connects Trainer and Profile. A 1-to-many relationship connects Trainer and Course.</p>	<pre>@startuml class Trainer { + listUpcomingClasses() } class Profile { + displayCourses() } class Course { } class CoursePage { - extendedDescription: String } class Member { } Trainer --> Member Trainer "1" -- "1" Profile @enduml</pre>	<pre> classDiagram class Trainer { + listUpcomingClasses() } class Profile { + displayCourses() } class Course { } class CoursePage { - extendedDescription: String } class Member { } Trainer "1" -- "1" Profile Trainer --> Member </pre> <p>This diagram includes the original classes and their methods. It also adds the listUpcomingClasses method to the Trainer class, the displayCourses method to the Profile class, and the extendedDescription attribute to the CoursePage class. The Trainer to Profile association is marked with a dashed line, indicating it is a derived association.</p>

			<p>Trainer "1" -- "*" Course Profile ..> Course Course "1" -- "1" CoursePage @enduml</p>	
HU-CL09	<pre>@startuml class CertificationCourse { courseId : Int title : String startDate : Date isOpen : Boolean } class CourseCatalog CourseCatalog ..> CertificationCourse : lists > @enduml</pre>	<pre> classDiagram class CourseCatalog class CertificationCourse { courseId : Int title : String startDate : Date isOpen : Boolean } CourseCatalog "1" *-- "*" CertificationCourse : lists </pre> <p>The diagram shows a class hierarchy. At the top is a class box labeled 'CourseCatalog'. Below it is a class box labeled 'CertificationCourse' with attributes: courseId (Int), title (String), startDate (Date), and isOpen (Boolean). A dashed line with a hollow arrowhead connects 'CourseCatalog' to 'CertificationCourse', labeled 'lists'.</p>	<pre> @startuml class Visitor { + viewUpcomingCourses(): List<CertificationCourse> } class Course { - name: String - description: String - schedule: Date } class CertificationCourse { } CertificationCourse --> Course Visitor ..> CertificationCourse @enduml </pre> <p>The code defines a 'Visitor' class with a method 'viewUpcomingCourses()' returning a list of 'CertificationCourse'. It also defines a 'Course' class with attributes: name (String), description (String), and schedule (Date). A class 'CertificationCourse' is defined. A directed association with multiplicity '1' to '1' connects 'CertificationCourse' to 'Course'. A generalization relationship with multiplicity '..>' connects 'Visitor' to 'CertificationCourse'.</p>	<pre> classDiagram class Visitor class CertificationCourse class Course Visitor ..> CertificationCourse CertificationCourse --> Course </pre> <p>This diagram illustrates the Visitor pattern. It shows a 'Visitor' class with a method 'viewUpcomingCourses()' returning a list of 'CertificationCourse'. Below it is a 'CertificationCourse' class. Further down is a 'Course' class with attributes: name (String), description (String), and schedule (Date). A generalization relationship with multiplicity '..>' connects 'Visitor' to 'CertificationCourse'. A directed association with multiplicity '1' to '1' connects 'CertificationCourse' to 'Course'.</p>
HU-CL10	<pre>@startuml class Event { eventId : Int name : String eventDate : Date location : String } class EventCatalog EventCatalog ..> Event : lists > @enduml </pre>	<pre> classDiagram class EventCatalog class Event { eventId : Int name : String eventDate : Date location : String } EventCatalog "1" *-- "*" Event : lists </pre> <p>The diagram shows a class hierarchy. At the top is a class box labeled 'EventCatalog'. Below it is a class box labeled 'Event' with attributes: eventId (Int), name (String), eventDate (Date), and location (String). A dashed line with a hollow arrowhead connects 'EventCatalog' to 'Event', labeled 'lists'.</p>	<pre> @startuml class Visitor { + viewUpcomingEvents(): List<Event> } class Event { - name: String - description: String - schedule: DateTime } Visitor ..> Event @enduml </pre> <p>The code defines a 'Visitor' class with a method 'viewUpcomingEvents()' returning a list of 'Event'. It also defines an 'Event' class with attributes: name (String), description (String), and schedule (DateTime). A generalization relationship with multiplicity '..>' connects 'Visitor' to 'Event'.</p>	<pre> classDiagram class Visitor class Event Visitor ..> Event </pre> <p>This diagram illustrates the Visitor pattern. It shows a 'Visitor' class with a method 'viewUpcomingEvents()' returning a list of 'Event'. Below it is an 'Event' class with attributes: name (String), description (String), and schedule (DateTime). A generalization relationship with multiplicity '..>' connects 'Visitor' to 'Event'.</p>