**Task 6**

**ternary search:**

**is a**[**decrease(by constant) and conquer algorithm**](http://www.geeksforgeeks.org/divide-and-conquer-introduction/)**that can be used to find an element in an**[**array**](https://www.geeksforgeeks.org/array-data-structure/)**. It is similar to**[**binary search**](http://www.geeksforgeeks.org/binary-search/)**where we divide the array into two parts but in this algorithm, we divide the given array into three parts and determine which has the key (searched element). We can divide the array into three parts by taking mid1 and mid2 which can be calculated as shown below. Initially, l and r will be equal to 0 and n-1 respectively, where n is the length of the array.**

**It is same as the binary search. The only difference is that, it reduces the time complexity a bit more. Its time complexity is O(log n base 3) and that of binary search is O(log n base 2).**

***mid1 = l + (r-l)/3***

**# Python3 program to illustrate**

**# recursive approach to ternary search**

**import math as mt**

**# Function to perform Ternary Search**

**def ternarySearch(l, r, key, ar):**

**if (r >= l):**

**# Find the mid1 and mid2**

**mid1 = l + (r - l) //3**

**mid2 = r - (r - l) //3**

**# Check if key is present at any mid**

**if (ar[mid1] == key):**

**return mid1**

**if (ar[mid2] == key):**

**return mid2**

**# Since key is not present at mid,**

**# check in which region it is present**

**# then repeat the Search operation**

**# in that region**

**if (key < ar[mid1]):**

**# The key lies in between l and mid1**

**return ternarySearch(l, mid1 - 1, key, ar)**

**elif (key > ar[mid2]):**

**# The key lies in between mid2 and r**

**return ternarySearch(mid2 + 1, r, key, ar)**

**else:**

**# The key lies in between mid1 and mid2**

**return ternarySearch(mid1 + 1,**

**mid2 - 1, key, ar)**

**# Key not found**

**return -1**

**# Driver code**

**l, r, p = 0, 9, 5**

**# Get the array**

**# Sort the array if not sorted**

**ar = [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]**

**# Starting index**

**l = 0**

**# length of array**

**r = 9**

**# Checking for 5**

**# Key to be searched in the array**

**key = 5**

**# Search the key using ternarySearch**

**p = ternarySearch(l, r, key, ar)**

**# Print the result**

**print("Index of", key, "is", p)**

**# Checking for 50**

**# Key to be searched in the array**

**key = 50**

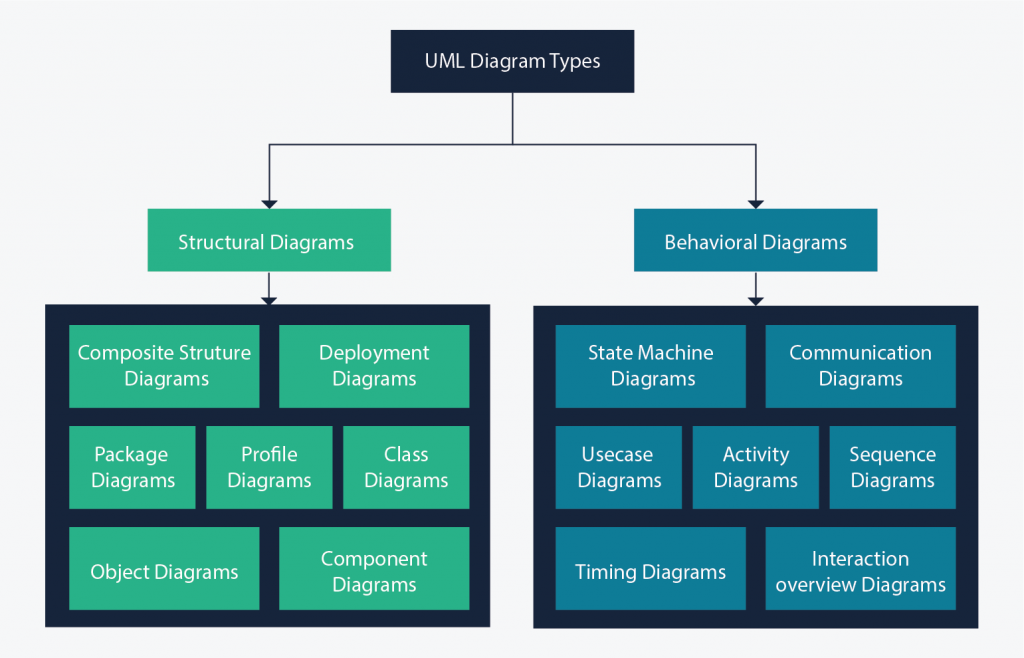
**# Search the key using ternarySearch**

**p = ternarySearch(l, r, key, ar)**

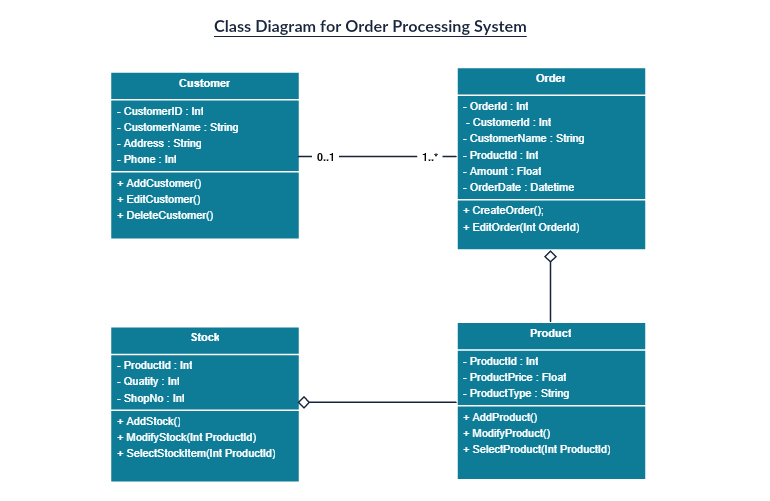
**# Print the result**

**print("Index of", key, "is", p)**

**UML Diagram:**

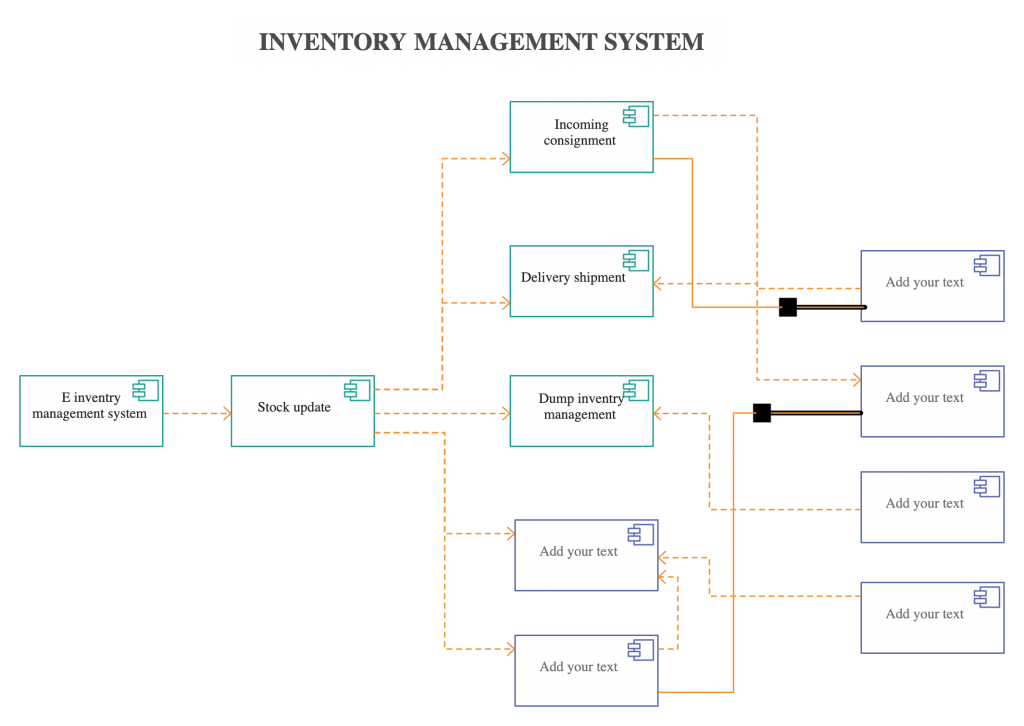
**UML stands for Unified Modeling Language. It’s a rich language to model software solutions, application structures, system behavior and**[**business processes**](https://creately.com/blog/diagrams/importance-of-business-process-modeling/)**. There are 14 UML diagram types to help you model these behaviors.** 

**class diagram: has three parts. Name at the top, attributes in the middle and operations or methods at the bottom. In a large system with many related classes, classes are grouped together to create class diagrams. Different relationships**[**between classes**](https://creately.com/diagram-type/article/relationships-that-exist-between-classes)**are shown by different types of arrows.**

****

**Component diagram:**

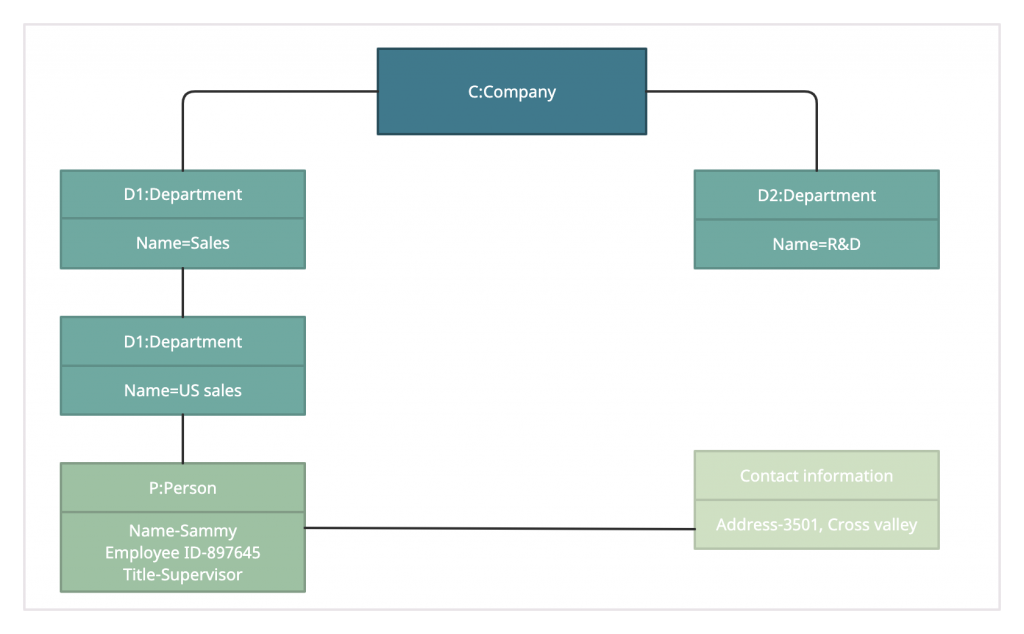
**A**[**component diagram**](https://creately.com/diagram-community/popular/t/component-diagram)**displays the structural relationship of components of a software system. These are mostly used when working with complex systems with many components. Components communicate with each other using**[**interfaces**](http://en.wikipedia.org/wiki/Interface_(object-oriented_programming))**. The interfaces are linked using connectors.**

****

**Object Diagram**

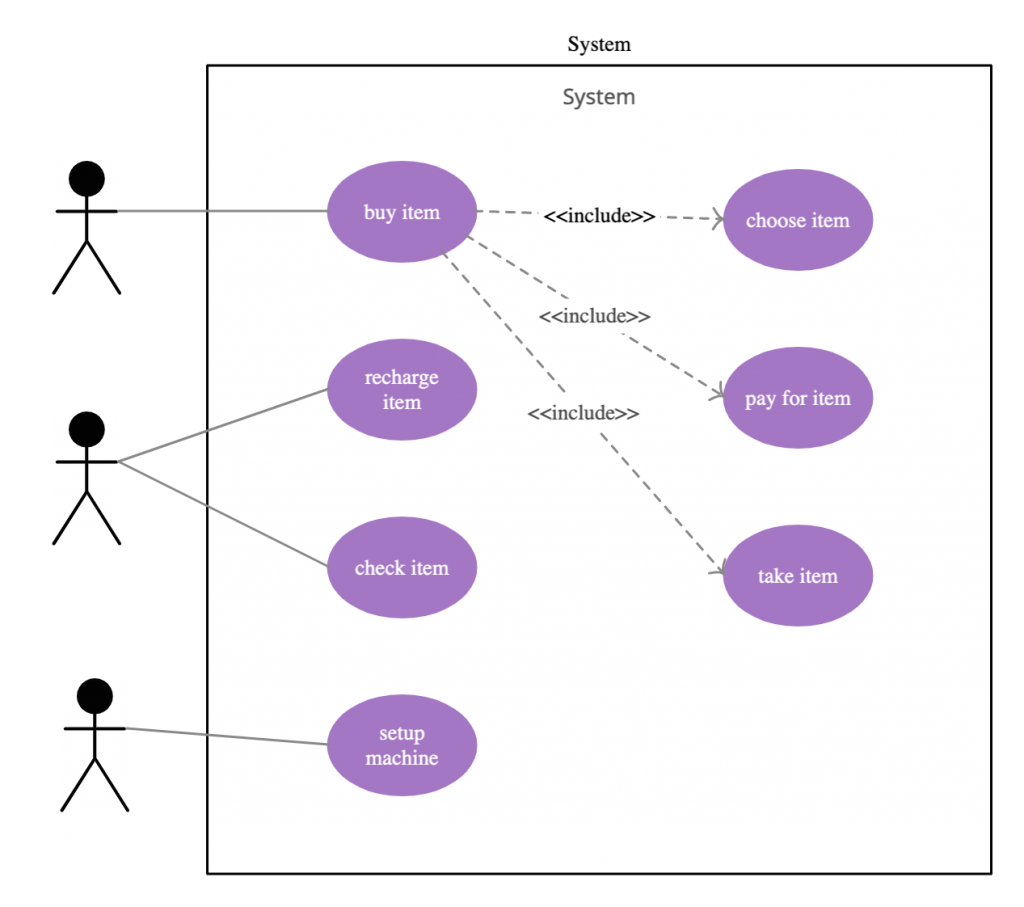
[**Object Diagrams**](https://creately.com/lp/object-diagram-tool/)**, sometimes referred to as Instance diagrams are very similar to**[**class diagrams**](https://creately.com/blog/diagrams/class-diagram-tutorial/)**. Like class diagrams, they also show the relationship between objects but they use real-world examples.**

**They show what a system will look like at a given time. Because there is data available in the objects, they are used to explain complex relationships between objects.**

****

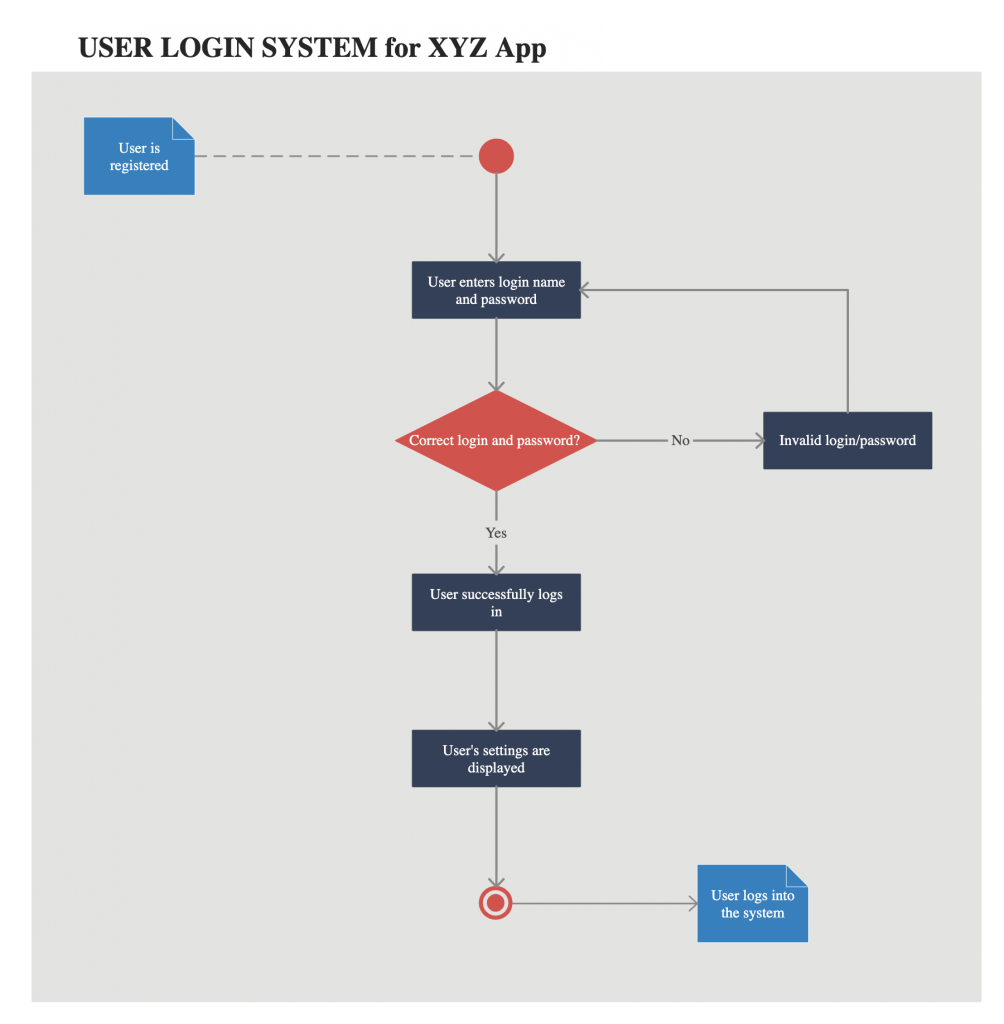
**Use Case Diagram**

**As the most known**[**diagram type**](https://creately.com/blog/diagrams/which-diagram-to-use/)**of the behavioral UML types,**[**Use case diagrams**](https://creately.com/diagram-community/popular/t/use-case)**give a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions interact.**

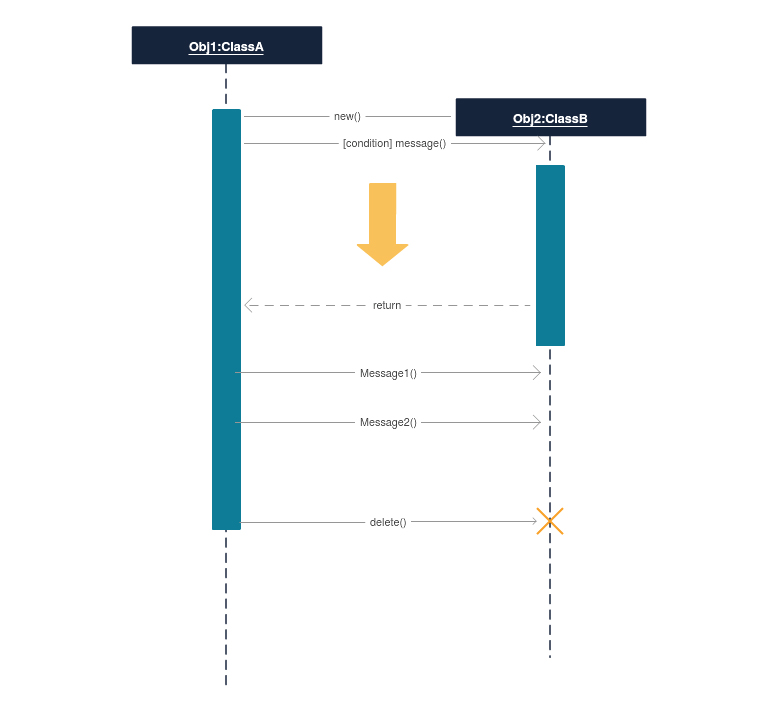
**It’s a great starting point for any project discussion because you can easily identify the main actors involved and the main processes of the system. **

**Activity Diagram**

**Activity diagrams represent workflows in a graphical way. They can be used to describe the business workflow or the operational workflow of any component in a system. Sometimes**[**activity diagrams**](https://creately.com/diagram-community/popular/t/activity-diagram)**are used as an alternative to State machine diagrams.**

****

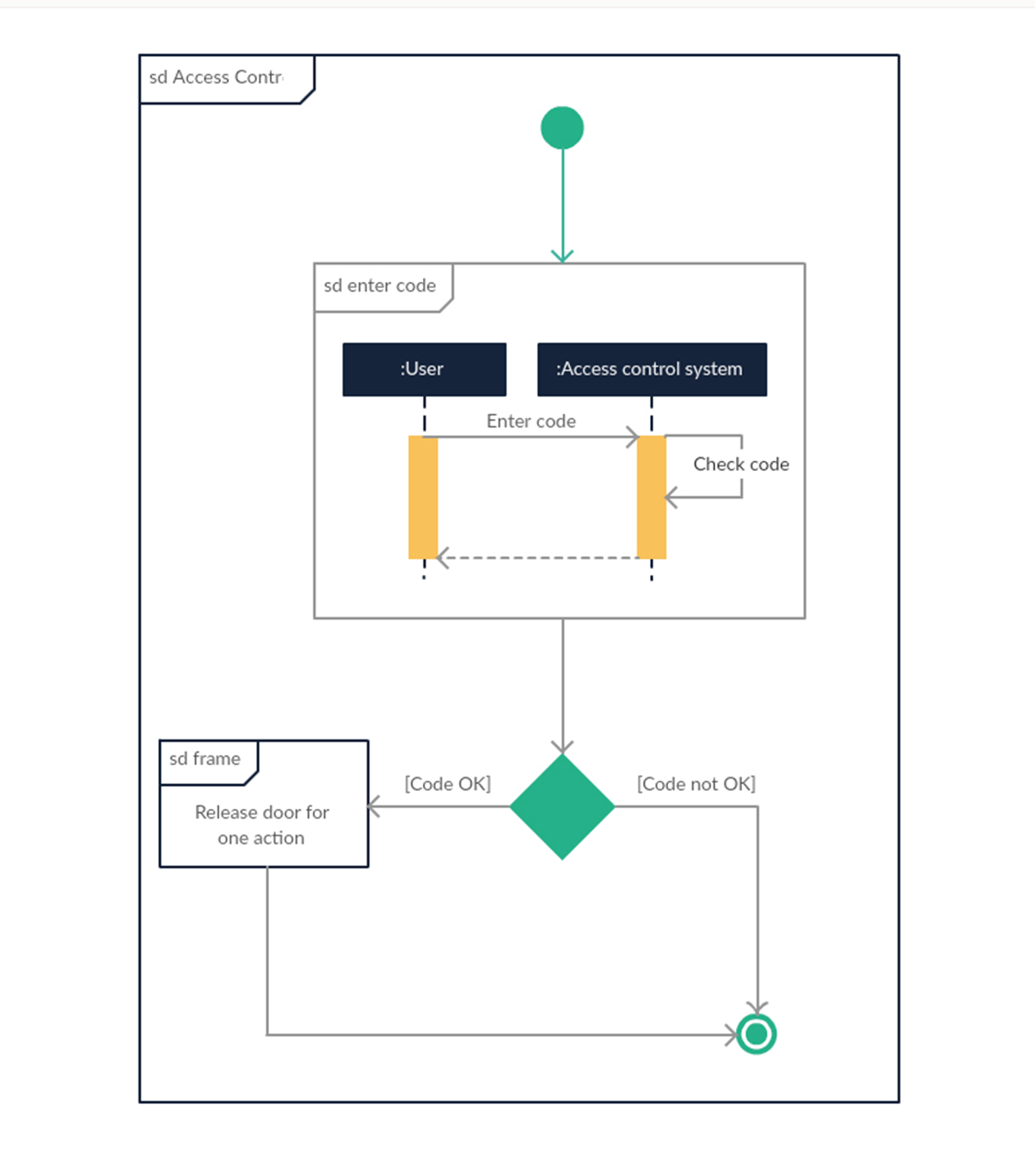
**Sequence Diagram**

[**Sequence diagrams**](https://creately.com/diagram-community/popular/t/sequence-diagram)**in**[**UML**](https://creately.com/lp/uml-diagram-tool)**show how objects interact with each other and the order those interactions occur. It’s important to note that they show the interactions for a particular scenario. The processes are represented vertically and interactions are shown as arrows. **

**Interaction Overview Diagram:**

**Interaction overview diagrams are very similar to activity diagrams. While activity diagrams show a sequence of processes, Interaction**[**overview diagrams**](https://creately.com/diagram/example/hoz91qr31/Interaction%20Overview%20Diagram)**show a sequence of interaction diagrams.**

**They are a collection of interaction diagrams and the order they happen. As mentioned before, there are seven types of interaction diagrams, so any one of them can be a node in an**[**interaction overview diagram**](https://creately.com/diagram/example/jk0ehse02/Interaction%20Overview%20Diagram)**.**

   
**Clean code :is focused —Each function, each class, each module exposes a single-minded attitude that remains entirely undistracted, and unpolluted, by the surrounding details**. *This is bad*

fna = 'Bob'

cre\_tmstp = 1621535852

*# This is good*

first name = 'Bob'

creation timestamp = 1621535852

Common design pattern: n

software engineering, a software design pattern is a general, reusable solution of how to solve a common problem when designing an application or system. Unlike a library or framework, which can be inserted and used right away, a design pattern is more of a template to approach the problem at hand.

Design patterns are used to support object-oriented programming (OOP), a paradigm that is based on the concepts of both objects (instances of a class; data with unique attributes) and classes (user-defined types of data).



