Main Task : Make sure app does what it's suppose to do.

Enums are great for type safety and value safety.

No dublicate Code!

If a object has a unused properties. Something wrong.

Object should do what their names indicate.

Each object should represent A SINGLE CONCEPT.

Seperating guiter spec from the guiter is encapsulation.

Encapsulation here doesn't mean private variables or methods,

Breaking app into LOGICAL PARTS also encupsulation.

Anytime you see dublicate code, look for encupsulation!

Flexibility -> keeping application from being fragile, no hard coding.

Encapsulation => Making code easy to change by encapsulating it to logical parts.

Functionality -> Make sure code does what it suppose to do.

Dessign Pattern => All about Reusability. Use a tested solution. Don't reinvent the wheel.

Delegation -> giving another object the responsibility of handling a particular task.

! Find the parts of your program that change often, and seperate the unchanged parts from it.

- SLAYT 2 -

1. Make sure your software does what the customer wants it to do.

Requirement: A requirement is a singular need detailing what a particular product or service should be or do. (System engineering, Software engineering).

Listen to the customers: Creating a requirement list. Customer doesn't know what they want.

Write Use Case: Steps that you system need to acomplish a particulr goal. Each use case provides one or more scenario.

Clear Value: Each use case have a clear value to system. Use case should help the customer to achive their goal.

Start and stop: Every use case must have a start point and stop point.

External initiators: Every use case start by external initiater, outside of system. Dog is it for dogdoor.

Start condition: Always in the first step, starts the action.

Stop condition: Determine the place that use ccase is finished.

Main Path: How a system works when everything is right.

Use case has a single goal, but can have multiple paths.

-SLAYT 3-

Requirements always change.

In use case flow, we should either X or X.1 not both of them.
Ex. 2- Todd hears barking, 2.1- bark recognizer hears barking.

The main path should be what you want to happen most of the time.

Any time you change your Use case. you need to check requirement list is it accomplish the requirement.

When your system needs to work in new or different way, begin by updating your use case,

Make sure your requirements cover all the steps in the use case.

-SLAYT 4-

- Delegation shields your objects from implementation changes to other objects in your software.

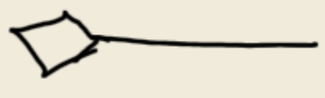
- The nouns in the use case are usually the classes you need.
- Looking at the nouns (and verbs) in your use case to figure out classes and methods is called textual analysis.
→ The verbs in your use case are usually the methods of the objects.

→ Why use Class Diagram:
→ helps you see the big picture and correct your mistakes.
→ helps you to explain your ideas to your boss and colleagues.
→ Multiplicity: allowed Barks: Bark[*] → these braced means there are more than one Bark.
"*" mean unlimited

-Slayt 5.1-

The abstract class defines behavior, and the subclasses implement that behavior.

whenever you find common behavior in two or more places, look to abstract that behavior into a class.

Aggregation:  One thing is <u>partly</u> made up of another thing.

Generalization : <u>inheritance</u>

Association: Relationship.

To see if a software well-designed, try and <u>change</u> it.

Coding to an interface rather than to an implementation, makes your software easier to extend.

Coding interfaces makes code more flexible.

Anytime you have behavior that you think is likely to change, you should always try to <u>encapsulate what varies</u>.

When you see a class that has more than one reason to change, it is probably <u>trying to do too many things.</u>

Each individual class does only one thing.

- SLAYT 5.2 -

Abstract classes are placeholders for actual implementation classes.

5.2 'nin 13. slaytına iyi bakın !

Cohesive Class : Does one thing really well and does not try to do something else.

Cohesion : Measures the degree of connectivity among the elements of a single module, class or object. Higher cohesion means well-defined and related.

→ Well-desiged software is easy to change and extend.

→ If a design isn't flexible then CHANGE IT.

→ Make sure each of your classes focus on doing ONE THING.

→ Encapsulate what varies.

→ Code to an interface rather than to an implementation.

→ Each class should have ONE reason to change

→ Classes are about Behavior and functionality.

Encapsulation,

Dizayn problemi bulma,

Fill in the blanks 'lere dikkat.