Rectangle

- **length:** *double*
  The length of one side of the rectangle
- **width:** *double*
  The width of the other side of the rectangle
+ **Rectangle**(length: *double*, width: *double*)
  Constructor that takes the length and width parameters
+ **setLength**(length: *double*): *void*
  A method that sets the length of the rectangle to the specified value, ensuring it's positive
+ **setWidth**(width: *double*): *void*
  A method that sets the width of the rectangle to the specified value, ensuring it's positive
+ **getArea**(): *double*
  A method that calculates and returns the area of the rectangle
+ **getPerimeter**(): *double*
  A method that calculates and returns the perimeter of the rectangle
+ **toString**(): *String*
  A method that returns a string representation of the rectangle indicating its length and width

Car

- **model**: *String*
  The model of the car
- **color**: *String*
  The color of the car
- **speed**: *int*
  The current speed of the car
- **isOn**: *boolean*
  Whether the car is currently turned on or off
- **acceleration**: *int*
  The rate at which the car accelerates
+ **Car**(model: *String*, color: *String*)
  Constructor that takes the model and color parameters
+ **Car**(model: *String*, color: *String*, acceleration: *int*)
  Constructor that takes the model, color, and acceleration parameters
+ **setAcceleration**(acceleration: *int*): *void*
  A method that sets the acceleration of the car to the specified value
+ **start**(): *void*
  A method that starts the car if it's not already running, accelerating it
+ **stop**(): *void*
  A method that stops the car if it's running, setting its speed to zero
+ **accelerate**(): *void*

A method that accelerates the car if it's running, increasing its speed by the acceleration rate

+ **accelerate**(acceleration: *int*): *void*
  A method that sets the acceleration of the car to the specified value and then accelerates the car
+ **toString**(): *String*
  A method that returns a string representation of the car indicating its color, model, and current speed if it's running

Person

- **name**: *String*
  The name of the person
- **car**: *Car*
  The car associated with the person, indicating that a person can use a car
- **wallet**: *Wallet*
  The wallet of the person, indicating aggregation where a person has a wallet, but both can exist separately
- **brain**: *Brain*
  The brain of the person, indicating composition where a person has a brain, and both cannot exist separately
+ **Person**(name: *String*)
  Constructor that takes the name parameter and initializes the brain
+ **Person**(name: *String*, car: *Car*, wallet: *Wallet*)
  Constructor that takes the name, car, and wallet parameters and initializes the brain
+ **getName**(): *String*
  A method that returns the name of the person
+ **think**(): *boolean*
  A method that simulates the thinking process of the person, returns true if something is remembered
+ **drive**(): *void*
  A method that simulates the process of driving where the person drives the associated car if available, accelerating if something is remembered during the thinking process
+ **buy**(item: *Item*): *void*
  A method that simulates the process of buying an item where the person buys the item if they have a wallet, deducting the item's value from the wallet and simulating the thinking process

Wallet

- **color**: *String*
  The color of the wallet
- **amount**: *int*
  The amount of money in the wallet
+ **Wallet**(color: *String*)
  Constructor that takes the color parameter

+ **Wallet**(color: *String*, amount: *int*)
    Constructor that takes the color and amount parameters
+ **getColor**(): *String*
    A method that returns the color of the wallet
+ **getAmount**(): *int*
    A method that returns the amount of money in the wallet
+ **addMoney**(money: *int): void*
    A method that adds the specified amount of money to the wallet if the amount is positive
+ **removeMoney**(value: *int*): *void*
    A method that removes the specified amount of money from the wallet if there's enough money available, otherwise throws an IllegalArgumentException
+ **toString**(): *String*
    A method that returns a string representation of the wallet in the format "Wallet[color]"

Brain

- **size**: *String*
    The size of the brain
- **person**: *Person*
    The person associated with the brain
+ **Brain**(person: *Person*)
    Constructor that takes the person parameter and initializes the size to "medium"
+ **Brain**(person: *Person*, size: *String*)
    Constructor that takes the person and size parameters
+ **getSize**(): *String*
    A method that returns the size of the brain
+ **chance**(): *int*
    A method that calculates the chance of remembering based on the size of the brain
+ **remember**(): *boolean*
    A method that simulates the process of remembering by generating a random number and comparing it with the chance of remembering
+ **toString**(): *String*
    A method that returns a string representation of the brain indicating the person's name and the size of the brain

Item

- name: *String*
    The name of the item
- value: *int*
    The value of the item
+ **Item**(name: *String*, value: *int*)
    Constructor that takes the name and value parameters
+ **getName**(): *String*
    A method that returns the name of the item

+ **getValue**(): *int*
    A method that returns the value of the item
+ **toString**(): *String*
    A method that returns a string representation of the item in the format "name (value)"