workshop number 3

TEACHnology

powered by AIESEC

Lesson 3 Planning:

- Why use OOP?
- Building blocks of OOP
  - Classes
  - Objects
- OOP concepts
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism

OOP

Why is it extensively used?

- Well suited for building trivial and complex applications
- Allows re-use of code thereby increasing productivity
- New features can be easily built into the existing code
- Reduced production cost and maintenance cost

OOP

Building Blocks of OOP: Objects & Classes

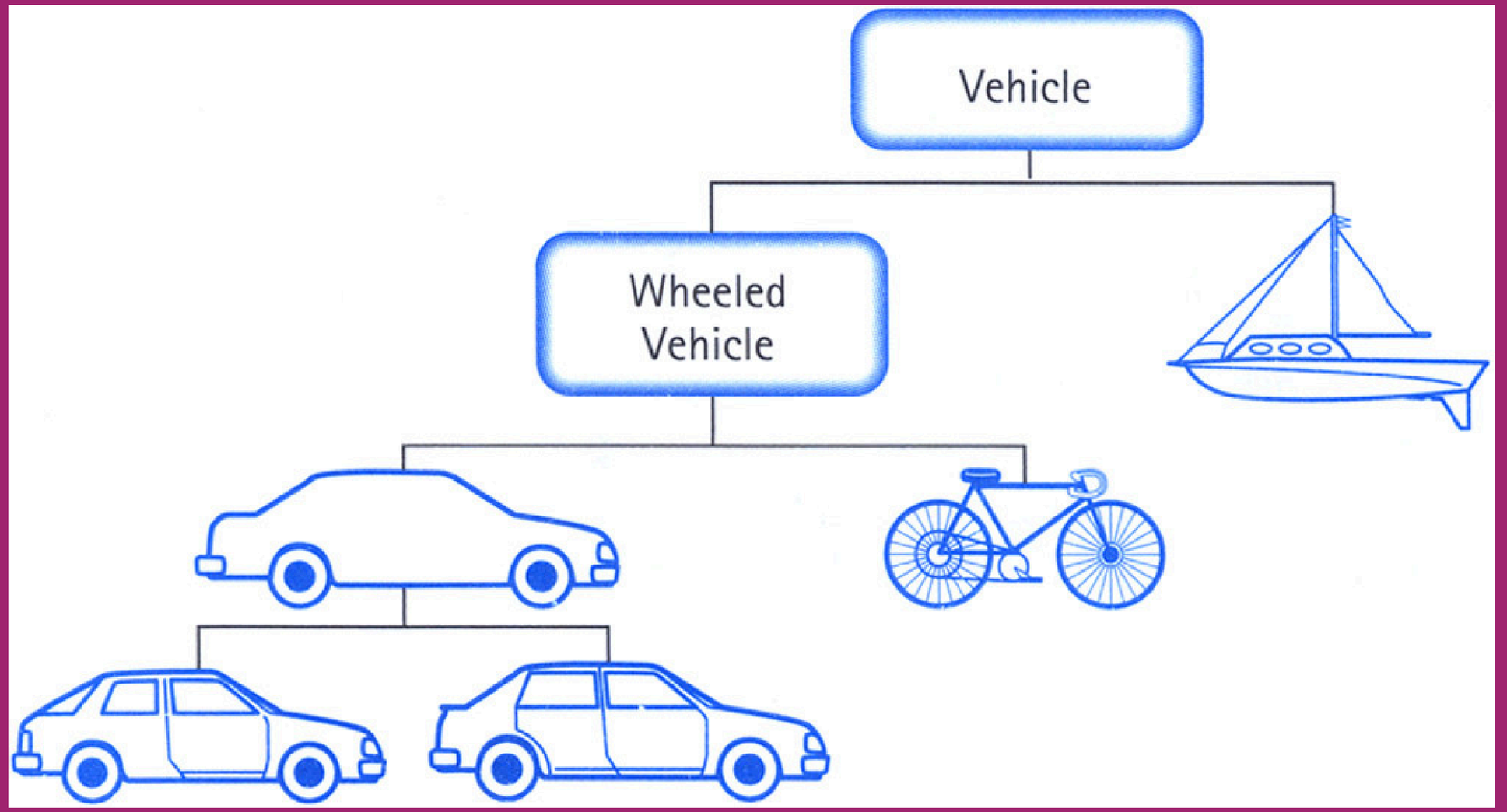
- Class has
  - Set of attributes or properties that describes every object
  - Set of behavior or actions that every object can perform

- Object has
  - Set of data (value for each of its attribute)
  - Set of actions that it can perform
  - An identity

Classes

- A class is a set of variables (to represent its attributes) and functions (to describe its behavior) that act on its variables

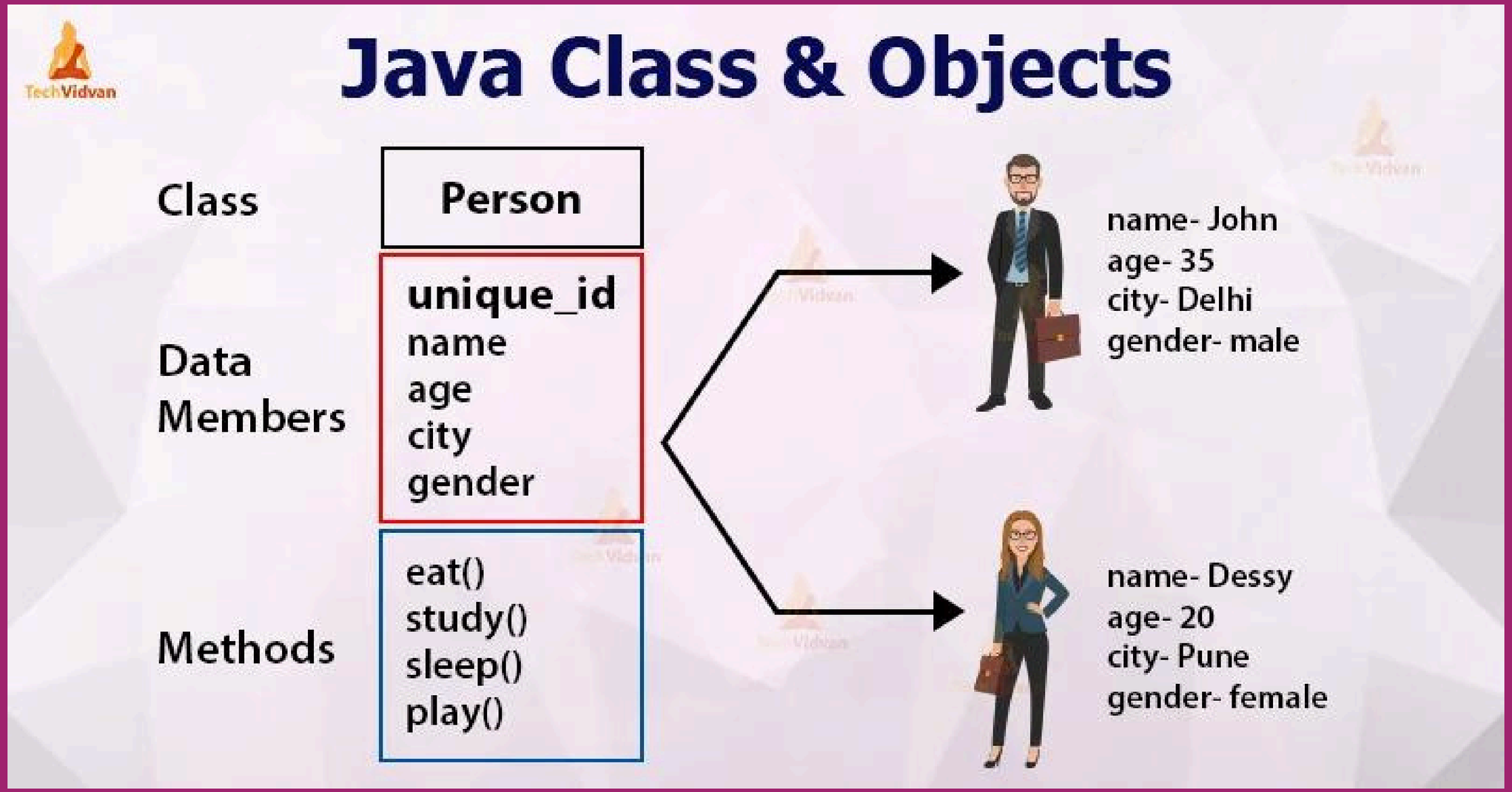
Classes



Objects

- Object is an instance of a class that holds data (values) in its variables. Data can be accessed by its functions

Classes


TechVidvan
Java Class & Objects
Class
Person
Data Members
unique_id
name
age
city
gender
Methods
eat()
study()
sleep()
play()
name- John
age- 35
city- Delhi
gender- male
name- Dessy
age- 20
city- Pune
gender- female

OOP concepts

OOP concepts

Abstraction

- Extracting essential properties and behavior of an entity
- Class represents such an abstraction and is commonly referred to as an abstract data type

Shipping Box

Abstraction

Attributes
Sender's name,
Receiver's name,
Cost of shipping per pound,
Weight
Behavior
Calculate shipping cost

Class

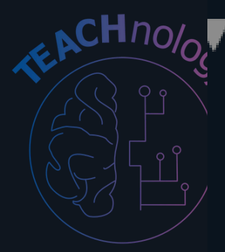
sender_name : string
receiver_name : string
cost_per_pound : int
weight : int

shipping_cost() : int

```
class MyClass{
  var MyHeight;
  var MyWeight;
  MyClass(this.MyHeight,this.MyWeight);
  double BMI(){
    return MyWeight/(MyHeight*MyHeight);



}
}
void main(){
  double Lenght=1.78;
  double Weight=60.0;
  MyClass MyBmi = MyClass(Lenght,Weight);
  print(MyBmi.BMI());
}
```

acces attributes within a class

```
class MyClass{
  var MyHeight;
  var MyWeight;
  MyClass(this.MyHeight,this.MyWeight);
  double BMI(){
    return MyWeight/(MyHeight*MyHeight);



}
}
void main(){
  double Lenght=1.78;
  double Weight=60.0;
  MyClass MyBmi = MyClass(Lenght,Weight);
  print(MyBmi.BMI());
}
```



Constructor

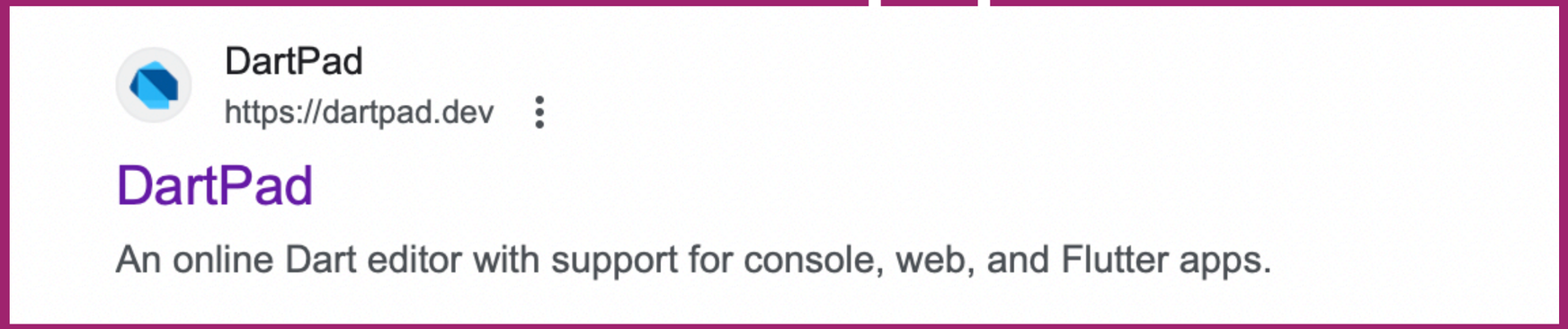
```
class MyClass{
  var MyHeight;
  var MyWeight;
  MyClass(this.MyHeight,this.MyWeight);
  double BMI(){
    return MyWeight/(MyHeight*MyHeight);



}
}
void main(){
  double Lenght=1.78;
  double Weight=60.0;
  MyClass MyBmi = MyClass(Lenght,Weight);
  print(MyBmi.BMI());
}
```

```dart
class MyClass{
  var MyHeight;
  var MyWeight;
  MyClass(this.MyHeight,this.MyWeight);
  double BMI(){
    return MyWeight/(MyHeight*MyHeight);


}
}
void main(){
  double Lenght=1.78;
  double Weight=60.0;
  MyClass MyBmi = MyClass(Lenght,Weight);
  print(MyBmi.BMI());
}
```

Creating an instance of a class

Exercise 1

Go to dartpap



Write a function that takes 2 int inputs and runs a math operation on them and returns a int

Exercise 2

write a class that has 3 attributes.

1st employee name.
2nd employee wage.
3rd if the said employee is older than 30 or not.
4th num of years the employee have been working

Exercise 2.5

Write a function inside the class that returns wage divided by num of years the employee have been working

OOP concepts

Encapsulation

- Mechanism by which we combine data and the functions that manipulate the data into one unit
- Objects & Classes enforce encapsulation

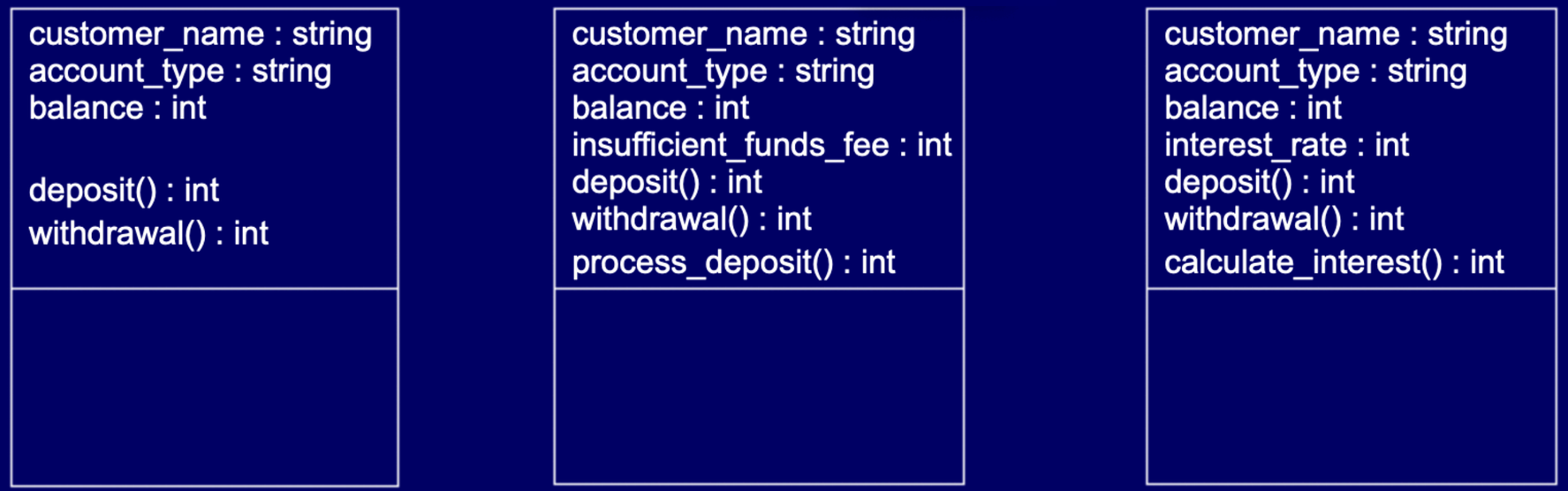
OOP concepts

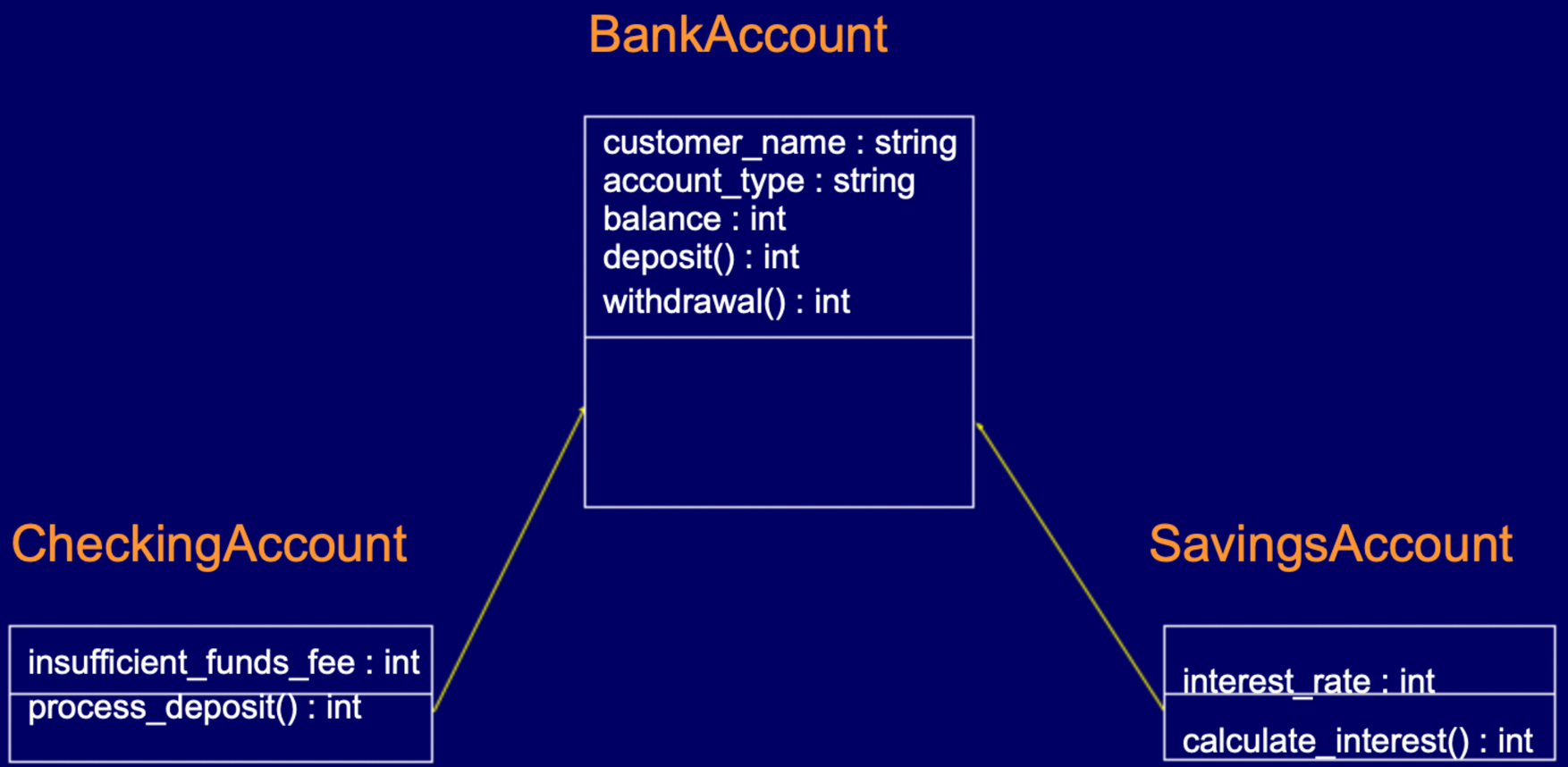
Inheritance

- Create new classes (derived classes) from existing classes (base classes)
- The derived class inherits the variables and functions of the base class and adds additional ones!
- Provides the ability to re-use existing code

Inheritance Example

BankAccount CheckingAccount SavingsAccount




BankAccount
customer_name : string
account_type : string
balance : int
deposit() : int
withdrawal() : int
CheckingAccount
insufficient_funds_fee : int
process_deposit() : int
SavingsAccount
interest_rate : int
calculate_interest() : int

Exercise3

1 -Create a base class called Person with the following attributes and methods:

- Attributes: name, age
- Methods: getDetails() -> prints the name and age of the person.

2 -Create a derived class called Student that inherits from Person with the following attributes and methods: // - Additional attribute:

- StudentId
- Additional method: getStudentId() -> prints the student ID

Exercise3.5

3 -Create another derived class called Teacher that inherits from Person with the following attributes and methods:

- Additional attribute: teacherId
- Additional method: getTeacherId() -> prints the teacher ID

4-Create objects for a student and a teacher, and demonstrate the inheritance and encapsulation concepts.

quiz

Go to gimkit.com/join and enter code

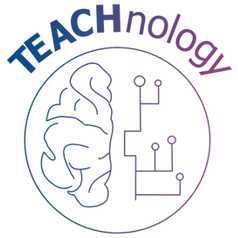
546279

```
1
2 ▼ int my_func(int a,int b){
3       return a+2*b;
4   }
5 ▼ main(){
6       int num1=10;
7       int num2=20;
8       print(my_func(num1,num2));
9   }
```

```
bool my_function(int num) {
  if(num%5==1){
    return true;
  }
  else{
    return false;
  }
}

void main(){
  int my_number=11;
  if(my_function(my_number)){
    print("my_number is cool");
   }
  else{
    print("my_number is not cool");
  }
}
```

```
class MyClass{
  var MyHeight;
  var MyWeight;
  MyClass(this.MyHeight,this.MyWeight);
  double BMI(){
    return MyWeight/(MyHeight*MyHeight);


}
}
void main(){
  double Lenght=1.78;
  double Weight=60.0;
  MyClass MyBmi = MyClass(Lenght,Weight);
  print(MyBmi.BMI());
}
```

```dart
class Car{
  int HP;
  int Doors;
  String Model;
  Car(this.HP,this.Doors,this.Model);

}

void main(){
  Car bmw = Car(5,2,'E3');
  Car toyota = Car(7,2,'E3');
  print(bmw.HP);
  bmw.HP=3;
  print(bmw.HP);
  print(toyota.HP);
```

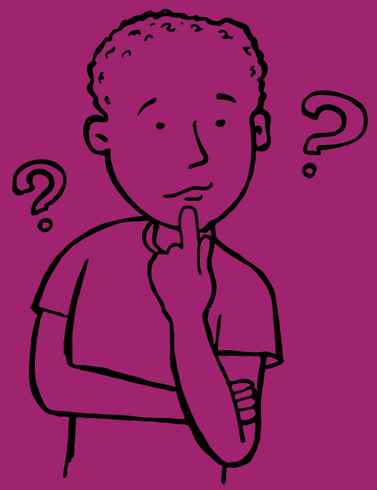


Widgets

- Stateless Widget
- Stateful Widget

Stateless Widgets

- The widgets whose state can not be altered once they are built
- A widget that does not require mutable state.



Now, if you got what I mean, tell me if these widgets are staleless or not?

Icons, Buttons, Text ...

Stateless Widgets

```
import 'package:flutter/material.dart';

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
        return Container();
}
}
```

Stateless Widgets


```dart
import 'package:flutter/material.dart';

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  // MyApp class declaration that extends StatelessWidget
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    // Build method that returns a widget and accepts a BuildContext as a parameter
    return Container();
    // Returns an empty Container widget
    // This is just a placeholder and can be replaced with actual UI components
  }
}
```

Stateless Widgets

BuildContext

- It's a special object that holds important information about the place of a widget in the Flutter app. It's like a map that tells the widget where it is and what's around it.
- Let's say you have a widget called "ProfilePicture" inside a widget called "UserCard." The BuildContext helps the "ProfilePicture" widget find its parent

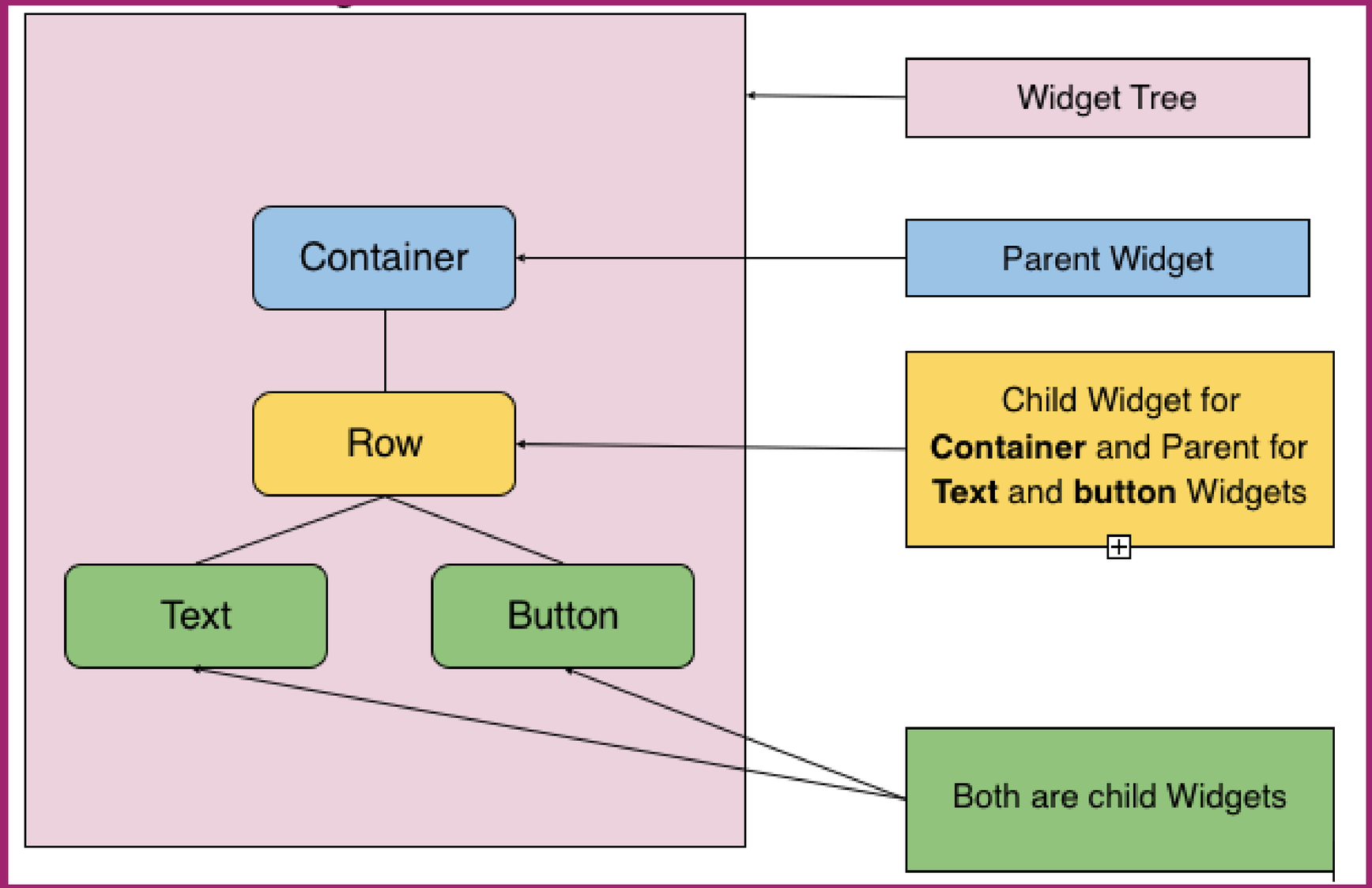
Stateless Widgets

BuildContext

summary

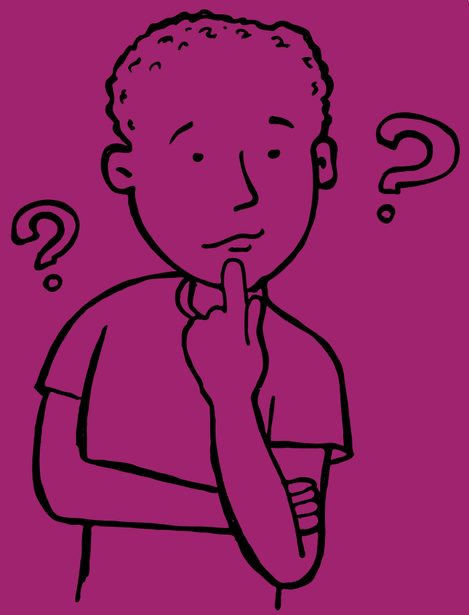
BuildContext is like a helpful guide for widgets in Flutter. It tells them where they are, helps them look good, find other widgets, and perform actions within the app.

Stateless Widgest



Stateful Widgets

- The widgets whose state can be altered once they are built
- A widget that has mutable state



Which one of these is not a stateful widget?

Form

TextField

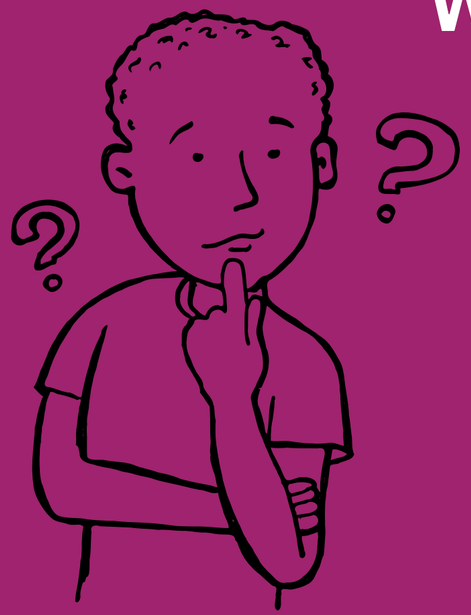
DropdownButton

Container

CheckboxListTile

Stateful Widgets

- The widgets whose state can be altered once they are built
- A widget that has mutable state



Which one of these is not a stateful widget?

Form

TextField

DropdownButton

Container

CheckboxListTile

Stateful Widget

```
class myHomePage extends StatefulWidget {
  const myHomePage({super.key});

  @override
  State<myHomePage> createState() => _myHomePageState();
}

class _myHomePageState extends State<myHomePage> {
  @override
  Widget build(BuildContext context) {
    return const Placeholder();
  }
}
```

Stateful Widget

```dart
import 'package:flutter/material.dart';

class myHomePage extends StatefulWidget {
  const myHomePage({Key? key}) : super(key: key); // Constructor for myHomePage widget

  @override
  State<myHomePage> createState() => _myHomePageState(); // Create state object for myHomePage
}

class _myHomePageState extends State<myHomePage> {
  @override
  Widget build(BuildContext context) {
    return const Placeholder(); // Return a Placeholder widget as the content of the home page
  }
}
```

BRAINSTORM

Come up With the idea for the app