## 1、类

- Getters 和 setters 方法

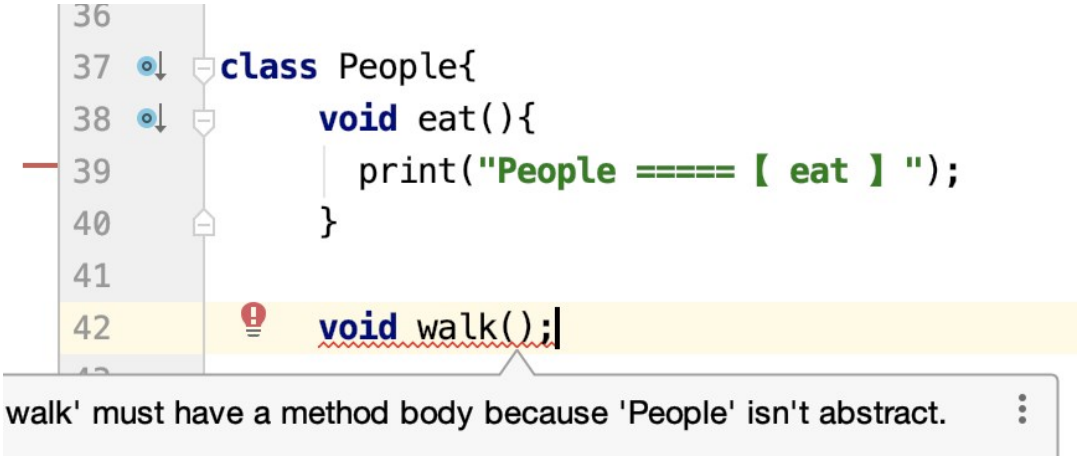
> 每个实例变量都有一个隐式的getter

```
class Rectangle {
  num left, top, width, height;
  Rectangle(num left,num top,num width,num height){
    this.left = left;
    this.top = top;
    this.width = width;
    this.height = height;
  }

  Rectangle(this.left, this.top, this.width, this.height);
  num get right => left + width;//=> 单行函数简单写法，返回值可有可无
  set right(num value) => left = value - width;

}
```

- 实现类中的方法，必须有body




- 抽象类中

```
abstract class People{
    void eat(){
      print("People =====【 eat 】");
    }
    void walk();
}
```

## 2、抽象类

除了不能实例化对象之外，类的其它功能依然存在，成员变量、成员方法和构造方法的访问方式和普通类一样。

**Java中抽象类表示的是一种继承关系，一个类只能继承一个抽象类，而一个类却可以实现多个接口。**

- 如果一个类包含抽象方法，那么该类必须是抽象类。
- 抽象类中不一定包含抽象方法，但是有抽象方法的类必定是抽象类。
- 抽象类中有成员变量、成员方法和构造方法 还可以有抽象方法

```
abstract  class People {
  String  name;
  People(String name) {
        }
        void eat() {
    System.out.println("People =====【 eat 】");
        }
        abstract void walk();
}
```

Dart 中抽象方法可以省略 abstract

```
abstract class People {
  String name;
  People({String name}) {
  }
  void eat() {
    print("People =====【 eat 】");
  }
  void walk();
}
```

- 任何子类必须重写父类的抽象方法，或者声明自身为抽象类

```
class Student extends People{
  @override
  void walk() {
     eat();
  }
}
```

Java

```
class Student extends People {
 Student(String name) {
               super(name);
               // TODO Auto-generated constructor stub
        }
        @Override
        void walk() {
               // TODO Auto-generated method stub
        }
}
```

dart 没有interface 关键字 但是有implements  每一个class (包括abstract class) 都隐含一个接口 都可以使用implement

```
abstract class Animal{
  void run();
}

 class People {
  String name;
  People({String name})
  void walk(){}
}
class Student implements People{
  @override
  String name;
  @override
  void walk() {
    // TODO: implement walk
  }
}
```

- 
- 任何子类必须重写父类的抽象方法，或者声明自身为抽象类。
- 
  - 抽象类不能被实例化(初学者很容易犯的错)，如果被实例化，就会报错，编译无法通过。只有抽象类的非抽象子类可以创建对象。
- 
  - 
- 
  - 抽象类中的抽象方法只是声明，不包含方法体，就是不给出方法的具体实现也就是方法的具体功能。、

    ```
      String name;

      People({String name}) {
      }
      void eat() {
        print("People =====[ eat ]");
      }
      void walk();
    }

    class Student extends People{
      @override
      void walk() {
         eat();
      }

    }
    ```

- 构造方法，类方法（用 static 修饰的方法）不能声明为抽象方法。

其他部分见技术分享ppt

## 3、mixin

mixin是在多个类层次结构中重用代码的一种方式。

```
mixin Animal{

  void run(){}
}

 mixin People {
  void walk(){}
}

class Student with Animal,People {

  void eat() {
    run();
    walk();
  }
}
```

## 4、泛型

- 泛型类

```
【abstract】class Cache<T> {
  T getByKey(String key);
  void setByKey(String key, T value);
}
```

- 泛型方法

  泛型参数()允许你在很多地方使用类型参数T

  - 在函数的返回中返回类型(T)
  - 在参数的类型中使用(List)
  - 在局部变量的类型中(T tmp)
  - 限制参数化类型

```
T first<T>(List<T> ts) {
  // Do some initial work or error checking, then...
  T tmp = ts[0];
  // Do some additional checking or processing...
  return tmp;
}
```

```
class Foo<T extends SomeBaseClass> {
  // Implementation goes here...
  String toString() => "Instance of 'Foo<$T>'";
}

class Extender extends SomeBaseClass {...}
```

可以使用SomeBaseClass 或它的任何子类作为泛型参数:

```
var someBaseClassFoo = Foo<SomeBaseClass>();
var extenderFoo = F
```

```
class GeneralTest<T>{
   T _vari;
   T get myVari{
     return _vari;
   }
   GeneralTest.newVV(){

   }

   GeneralTest(T t){
     this._vari = t;

   }

   void setV(T v){
     _vari = v;
   }

   static T newV<T>(T t){

     return  t;
   }

}

class Test{

  void main(){
    GeneralTest<BT>.newVV();//命名构造函数
//    GeneralTest.newV(BT(i: 2)).name; 静态方法
  }
}
```

对provider 的封装

```
import 'package:provide/provide.dart';
import 'package:xxwq_flutter/libs/states/config_state_model.dart';

class ProvierHelper{

  static var providers = Providers();
  static var themeConfigModel = ThemeConfigModel();

  static init({model,child,dispose = true}){

    providers = Providers()
      ..provide(Provider<ThemeConfigModel>.value(themeConfigModel));

    return ProviderNode(child: child, providers: providers,dispose: dispose);
  }

  static connect<T>({builder, child, scope}) {
    return Provide<T>(builder: builder, child: child, scope: scope);
  }
  static T value<T>(context,{scope}){
    return Provide.value<T>(context,scope: scope);
  }
}
```

分类: flutter

标签: flutter

好文要顶 关注我 收藏该文

肖无情
粉丝 - 2 关注 - 5
+加关注

0 推荐 0 反对

« 上一篇： 02-01dart语法
» 下一篇： 02-03 flutter异步