ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

Object-Oriented Programming

Abstract Class

Objectives

- Abstraction
- Abstract class
- Abstract method

Abstraction

- **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.
- Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing of the message delivery.
- Abstraction lets you focus on what the object does instead of how it does it.

Abstraction

- There are two ways to achieve abstraction in java:
  - Abstract class (0 to 100%)
  - Interface (100%)

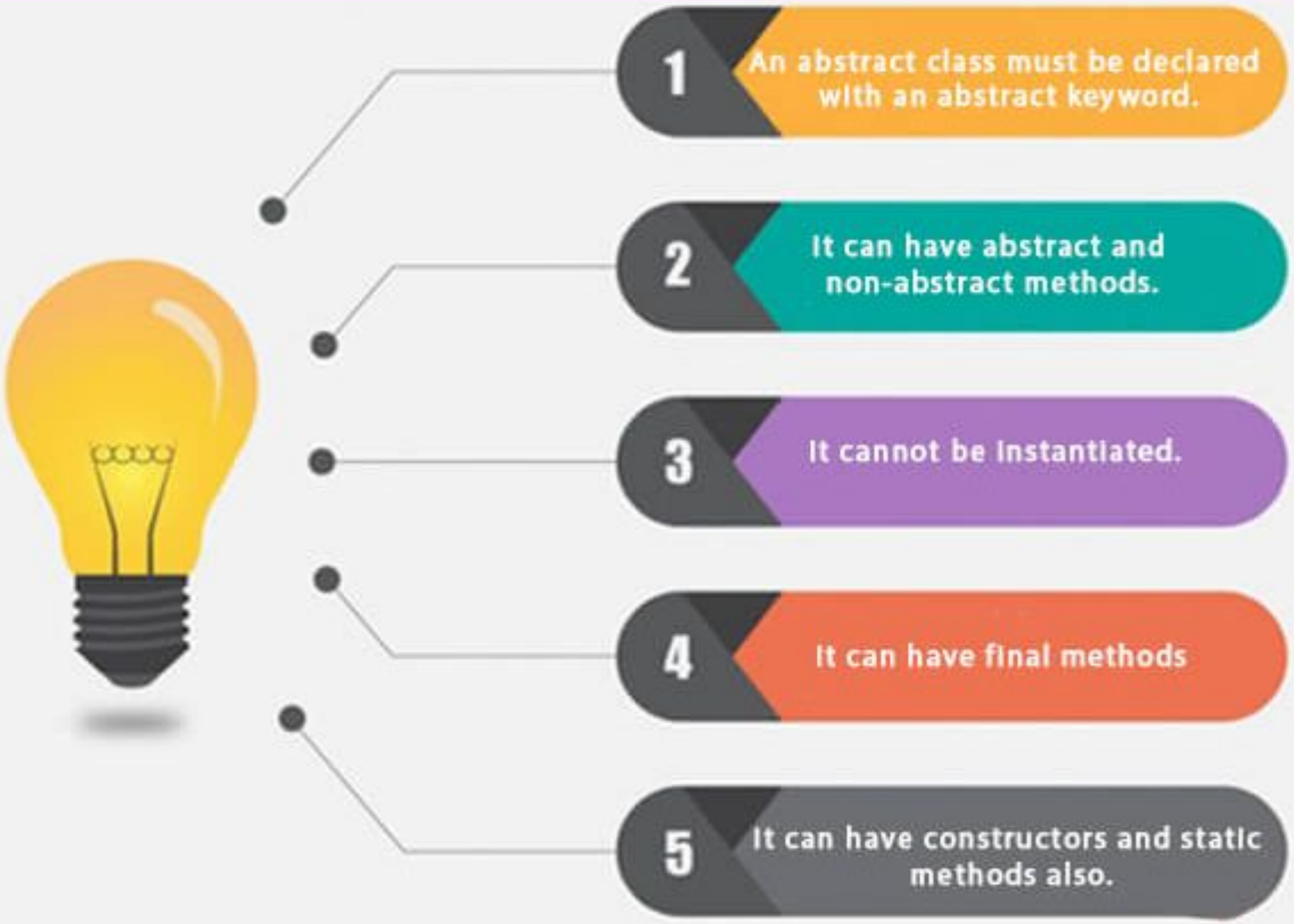
Abstract class in Java

- A class that is declared as abstract is known as an **abstract class**.
- It can have abstract and non-abstract methods.
- It needs to be extended and its method implemented.
- **It cannot be instantiated**.

Abstract class in Java

- An abstract class must be declared with an abstract keyword.
- It can have abstract and non-abstract methods.
- It cannot be instantiated.
- It can have constructors and static methods also.
- It can have final methods which will force the subclass not to change the body of the method.

Abstract class in Java



Abstract class in Java

- **Syntax**

```
abstract class A
{
    //attributes;
    //methods;
}
```

Abstract class in Java

- **Syntax**

```
abstract class A
{
    //attributes;
    //methods;
    abstract void printStatus();//no method body and abstract
}
```

Examples

```
abstract class Bike
{
    abstract void run();
}
class Honda4 extends Bike
{
    void run(){System.out.println("running safely");}
        public static void main(String args[])
        {
          Bike obj = new Honda4();
          obj.run();
    }
}
```

Examples

```
abstract class Shape{
abstract void draw();
}
//In real scenario, implementation is provided by others i.e. unknown by end user
class Rectangle extends Shape{
void draw(){System.out.println("drawing rectangle");}
}
class Circle1 extends Shape{
void draw(){System.out.println("drawing circle");}
}
//In real scenario, method is called by programmer or user
class TestAbstraction1{
public static void main(String args[]){
Shape s=new Circle1();//In a real scenario, object is provided through method, e.g., getShape() method
s.draw();
}
}
```

```
drawing circle
```

Examples

```
abstract class Bank{
abstract int getRateOfInterest();
}
class SBI extends Bank{
int getRateOfInterest(){return 7;}
}
class PNB extends Bank{
int getRateOfInterest(){return 8;}
}

class TestBank{
public static void main(String args[]){
Bank b;
b=new SBI();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
b=new PNB();
System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");
}}
```

```
Rate of Interest is: 7 %
Rate of Interest is: 8 %
```

Examples

```
abstract class Bike{
  Bike(){System.out.println("bike is created");}
  abstract void run();
  void changeGear(){System.out.println("gear changed");}
}
//Creating a Child class which inherits Abstract class
class Honda extends Bike{
void run(){System.out.println("running safely..");}
}
//Creating a Test class which calls abstract and non-abstract methods
class TestAbstraction2{
public static void main(String args[]){
 Bike obj = new Honda();
 obj.run();
 obj.changeGear();
}
}
```

```
bike is created
running safely..
gear changed
```

Examples

```
abstract class Bike{
  Bike(){System.out.println("bike is created");}
  abstract void run();
  void changeGear(){System.out.println("gear changed");}
}
//Creating a Child class which inherits Abstract class
class Honda extends Bike{
void run(){System.out.println("running safely..");}
}
//Creating a Test class which calls abstract and non-abstract methods
class TestAbstraction2{
public static void main(String args[]){
 Bike obj = new Honda();
 obj.run();
 obj.changeGear();
}
}
```

```
bike is created
running safely..
gear changed
```

Notations

- If there is an abstract method in a class, that class must be abstract.
- If you are extending an abstract class that has an abstract method, you must either provide the implementation of the method or make this class abstract.
- The abstract class can also be used to provide some implementation of the interface. In such case, the end user may not be forced to override all the methods of the interface.

Examples

```
interface A{
void a();
void b();
void c();
void d();
}

abstract class B implements A{
public void c(){System.out.println("I am c");}
}

class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}
```

```
class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

```
Output:I am a
       I am b
       I am c
       I am d
```