Abstract

- `abstract` keyword, does not have any body, the child class must override.
- the `abstract` parent class cannot be directly used, objects of `abstract` class cannot be created.
- any class that contains one or more `abstract` methods must also be declared as `abstract`.
- the best use of `abstract` is, when the child class would always override the parent class.

Parent Class

```
package com.inclass.abstractDemo;

public abstract class Parent {
    abstract void career(String name);
}
```

Child Class 1

```
package com.inclass.abstractDemo;

public class Son extends Parent {

    @Override
    void career(String name) {
```

```java
        System.out.println(name + " will be a
developer");
    }
}
```

Child Class 2

```java
package com.inclass.abstractDemo;

public class Daughter extends Parent {

    @Override
    void career(String name) {
        System.out.println(name + " will be a
doctor");
    }
}
```

Main

```java
package com.inclass.abstractDemo;

public class Main {
    public static void main (String[] args) {
        Son son = new Son();
        son.career("Boy");

        Daughter daughter = new Daughter();
        daughter.career("Girl");
    }
}
```

Output

Boy will be a developer
Girl will be a doctor

Parent Class

```java
package com.inclass.abstractDemo;

public abstract class Parent {
    abstract void career(String name);

    static void greeting() {
        System.out.println("Hello World !");
    }
}
```

Main

```java
package com.inclass.abstractDemo;

public class Main {
    public static void main (String[] args) {
        Parent.greeting();
    }
}
```

Output

Hello World !

- this works because `static` methods doesn't need objects, and objects of `abstract` class can be created.

Interfaces

- Interfaces = abstract + multiple inheritance

Interface 1

```
package com.inclass.interfaces;

public interface Engine {

    static final int PRICE = 78000;

    void start();
    void stop();
    void accelerate();
}
```

Interface 2

```
package com.inclass.interfaces;

public interface Brake {
    void brake();
}
```

Class

```
package com.inclass.interfaces;

public class Car implements Engine, Brake{

    @Override
    public void brake() {
        System.out.println("I brake like a normal 
car");
    }
    @Override
    public void start() {
        System.out.println("I start like a normal 
car");
    }
    @Override
    public void stop() {
        System.out.println("I stop like a normal 
car");
    }
    @Override
```

```java
    public void accelerate() {
        System.out.println("I accelerate like a normal car");
    }
}
```

Main

```java
package com.inclass.interfaces;

public class Main {
    public static void main(String[] args) {
        Car car = new Car();
        car.accelerate();
        car.start();
        car.stop();
    }
}
```

Output

```
I accelerate like a normal car
I start like a normal car
I stop like a normal car
```

- when we use classes, the parent class has no idea of child class, but the child class has idea about parent classes. when a function is called from child class and the same function exists in parent class, then the child class and parent class both has to be present at compile time.

- two class which are unrelated to each other can implement the same interfaces.

Important

Don't use Interfaces in performance critical code, as this is executed at runtime.

Default

- `default` allows interfaces to be extended, without any changes in code.
- `default` method does not always need to be implemented in class.
- But class implementation > default implementation

Interface 1

```java
package com.inclass.interfaces.default;

public interface A {
    default void fun() {
        System.out.println("A → fun");
    }
}
```

Interface 2

```java
package com.inclass.interfaces.default;

public interface B {
    void run();
}
```

Class

```java
package com.inclass.interfaces.default;

public class Main implements A, B {
    @Override
    public void run() {
        System.out.println("B → run");
    }

    public static void main(String[] args) {
        Main obj = new Main();
        obj.fun();
        obj.run();
    }
}
```

Output

```
A → fun
B → run
```

- `static` methods cannot be inherited, and it cannot be overridden, so `static` methods must have a body and must be called by interface name.

Class

```java
package com.inclass.interfaces.default;

public interface A {

    static void greeting() {
        System.out.println("Hello World !");
    }
}
```

Main

```java
package com.inclass.interfaces.default;

public class Main implements A, B {
    @Override
    public void run() {
        System.out.println("B → run");
    }

    public static void main(String[] args) {
        A.greeting();
    }
}
```

Output

Hello World !

Important

The access modifier in child class, must no be more restricting, than the parent class. (same for interfaces).

~~~
title: Parent Class
```java
package com.inclass.access;

public class C {
    String name;
    public C(String name) {
        this.name = name;
    }
    public void name() {
        System.out.println("I am " + name);
    }
}
```
~~~

~~~
title: Child Class with main
```java
package com.inclass.access;

public class D extends C{

    public D(String name) {
        super(name);
    }
    @Override
    protected void name() {
        System.out.println("I am not " + name);
    }    public static void main(String[] args) {
        D d = new D("Driptanil");
        d.name();
    }
}
```
````ad-error
```
java: name() in com.inclass.access.D cannot override
name() in com.inclass.access.C
  attempting to assign weaker access privileges; was
public
```
````
~~~

~~~
title: Parent Class
```java
package com.inclass.access;
~~~

~~~
public class C {
    String name;

    public C(String name) {
        this.name = name;
    }
    protected void name() {
        System.out.println("I am " + name);
    }
}
```
~~~

~~~
title: Child Class with main
```java
package com.inclass.access;

public class D extends C{

    public D(String name) {
        super(name);
    }
    @Override
    public void name() {
        System.out.println("I am not " + name);
    }
    public static void main(String[] args) {
        D d = new D("Driptanil");
        d.name();
    }
}
```
~~~

~~~
```ad-output
I am not Driptanil
```
~~~