Interfaces

Like many type systems, GraphQL supports interfaces. An **Interface** is an abstract type that includes a certain set of fields that a type must include to implement the interface (read more here).

Code first

When using the code first approach, you define a GraphQL interface by creating an abstract class annotated with the `@InterfaceType()` decorator exported from the `@nestjs/graphql`.

```
import { Field, ID, InterfaceType } from '@nestjs/graphql';

@InterfaceType()
export abstract class Character {
  @Field((type) => ID)
  id: string;

  @Field()
  name: string;
}
```

> warning **Warning** TypeScript interfaces cannot be used to define GraphQL interfaces.

This will result in generating the following part of the GraphQL schema in SDL:

```
interface Character {
  id: ID!
  name: String!
}
```

Now, to implement the `Character` interface, use the `implements` key:

```
@ObjectType({
  implements: () => [Character],
})
export class Human implements Character {
  id: string;
  name: string;
}
```

> info **Hint** The `@ObjectType()` decorator is exported from the `@nestjs/graphql` package.

The default `resolveType()` function generated by the library extracts the type based on the value returned from the resolver method. This means that you must return class instances (you cannot return literal JavaScript objects).

To provide a customized `resolveType()` function, pass the `resolveType` property to the options object passed into the `@InterfaceType()` decorator, as follows:

```
@InterfaceType({
  resolveType(book) {
    if (book.colors) {
      return ColoringBook;
    }
    return TextBook;
  },
})
export abstract class Book {
  @Field((type) => ID)
  id: string;

  @Field()
  title: string;
}
```

Interface resolvers

So far, using interfaces, you could only share field definitions with your objects. If you also want to share the actual field resolvers implementation, you can create a dedicated interface resolver, as follows:

```
import { Resolver, ResolveField, Parent, Info } from '@nestjs/graphql';

@Resolver(type => Character) // Reminder: Character is an interface
export class CharacterInterfaceResolver {
  @ResolveField(() => [Character])
  friends(
    @Parent() character, // Resolved object that implements Character
    @Info() { parentType }, // Type of the object that implements
Character
    @Args('search', { type: () => String }) searchTerm: string,
  ) {
    // Get character's friends
    return [];
  }
}
```

Now the `friends` field resolver is auto-registered for all object types that implement the `Character` interface.

Schema first

To define an interface in the schema first approach, simply create a GraphQL interface with SDL.

```
interface Character {
  id: ID!
  name: String!
}
```

Then, you can use the typings generation feature (as shown in the quick start chapter) to generate corresponding TypeScript definitions:

```
export interface Character {
  id: string;
  name: string;
}
```

Interfaces require an extra `__resolveType` field in the resolver map to determine which type the interface should resolve to. Let's create a `CharactersResolver` class and define the `__resolveType` method:

```
@Resolver('Character')
export class CharactersResolver {
  @ResolveField()
  __resolveType(value) {
    if ('age' in value) {
      return Person;
    }
    return null;
  }
}
```

> info **Hint** All decorators are exported from the `@nestjs/graphql` package.