



The Game with Ogres and Knights
 One day, we are interested code in a clash of a knight and ogre:
 a knight attacks an ogre,
 and the ogre makes a revenge afterwards
 In the attack by a knight, an ogre loses 10% of its energy
 If an ogre has more energy, a knight will lose 10% of his energy during his attack

How would you implement a clash of a knight and an ogre in C (procedurally)?

```
typedef struct {
  int energy;
} Ogre;
```

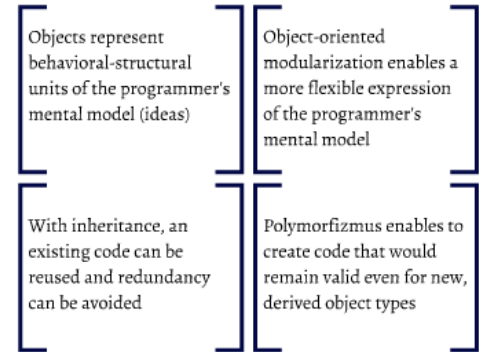
What's missing in the struct representation of the characters?

```
typedef struct {
  int energy;
  void (*revenge)(Knight k);
} Ogre;
```

Object:
 - identity
 - state
 - behavior
 Program:
 an interaction of objects

There are programming languages that enable to create programs exactly the way that is close to the programmer's mental model

A class:
 a prescription for the object creation



How would we in the structure implementation in C add a kind of a bad ogre who would, if hungry, during a revenge draw all the energy of the knight?

```
typedef enum {common, bad, timid} OgreKind;
typedef struct {
  int energy;
  OgreKind kind;
  bool hungry;
  void (*revenge)(Knight k);
} Ogre;
```

BadOgre inherits the behavior and structure from the Ogre class

BadOgre *extends* and *concretizes* the Ogre class: adds new details into the ogre abstraction

How would adding new kinds of ogres would exhibit in the *clash*?

Lecture 1:

Insight into Object-Oriented Programming

Valentino Vranić

**Ústav informatiky, informačných systémov a
softvérového inžinierstva**



vranic@stuba.sk

fiit.sk/~vranic

OOP 2019/20

18. 2. 2020

The Game with Ogres and Knights

For now, we are interested only in a clash of a knight and ogre:

a knight attacks an ogre,
and the ogre makes a revenge afterwards

In the attack by a knight, an ogre loses 10% of its energy

If an ogre has more energy, a knight will lose 10% of his energy during his attack

For now, we are interested only in a clash of a knight and ogre:

a knight attacks an ogre,
and the ogre makes a revenge afterwards

In the attack by a knight, an ogre loses 10% of its energy

If an ogre has more energy, a knight will lose 10% of his energy during his attack

How would you implement a clash of a knight and an ogre in C (procedurally)?

```
typedef struct {  
    int energy;  
}  
Ogre;
```

What's missing in
the struct
representation of
the characters?

```
typedef struct {  
    int energy;  
    void (*revenge)(Knight k);  
} Ogre;
```

Object:

- identity
- state
- behavior

Program:

an interaction of objects

Objects represent
behavioral-structural
units of the programmer's
mental model (ideas)

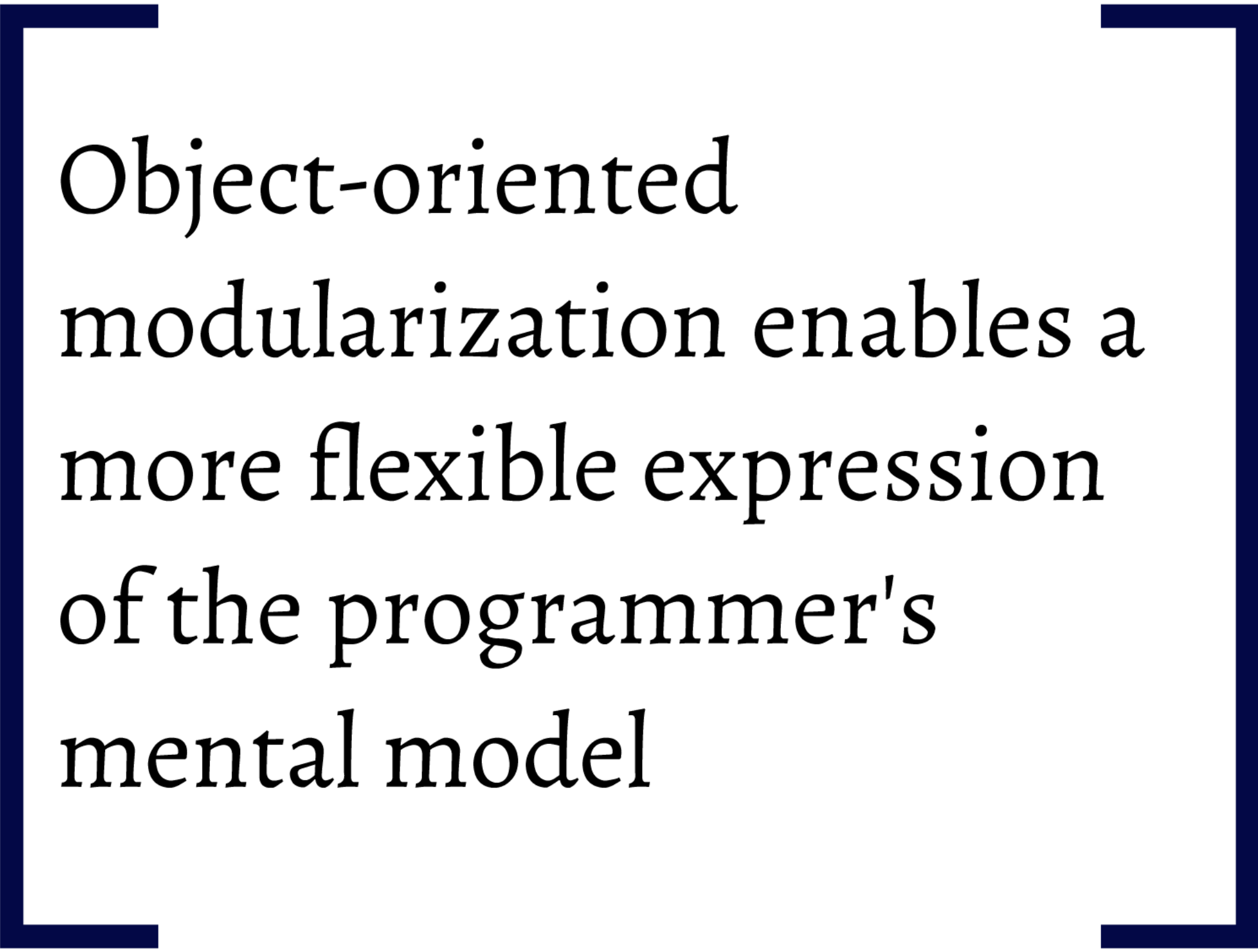
There are programming languages that enable to create programs exactly the way that is close to the programmer's mental model

A class:

a prescription

for the object

creation



Object-oriented
modularization enables a
more flexible expression
of the programmer's
mental model

How would we in the structure implementation in C add a kind of a bad ogre who would, if hungry, during a revenge draw all the energy of the knight?

```
typedef enum {common, bad, timid} OgreKind;
```

```
typedef struct {  
    int energy;  
    OgreKind kind;  
    bool hungry;  
    void (*revenge)(Knight k);  
} Ogre;
```

BadOgre inherits the behavior and structure from the Ogre class

BadOgre **extends** and **concretizes** the Ogre class: adds new details into the ogre abstraction

With inheritance, an existing code can be reused and redundancy can be avoided

How would adding new kinds of ogres would exhibit in the **clash**?

Polymorfizmus enables to
create code that would
remain valid even for new,
derived object types

Objects represent behavioral-structural units of the programmer's mental model (ideas)

Object-oriented modularization enables a more flexible expression of the programmer's mental model

With inheritance, an existing code can be reused and redundancy can be avoided

Polymorfizmus enables to create code that would remain valid even for new, derived object types

Objects represent behavioral-structural units of the programmer's mental model (ideas)

Object-oriented modularization enables a more flexible expression of the programmer's mental model

With inheritance, an existing code can be reused and redundancy can be avoided

Polymorfizmus enables to create code that would remain valid even for new, derived object types