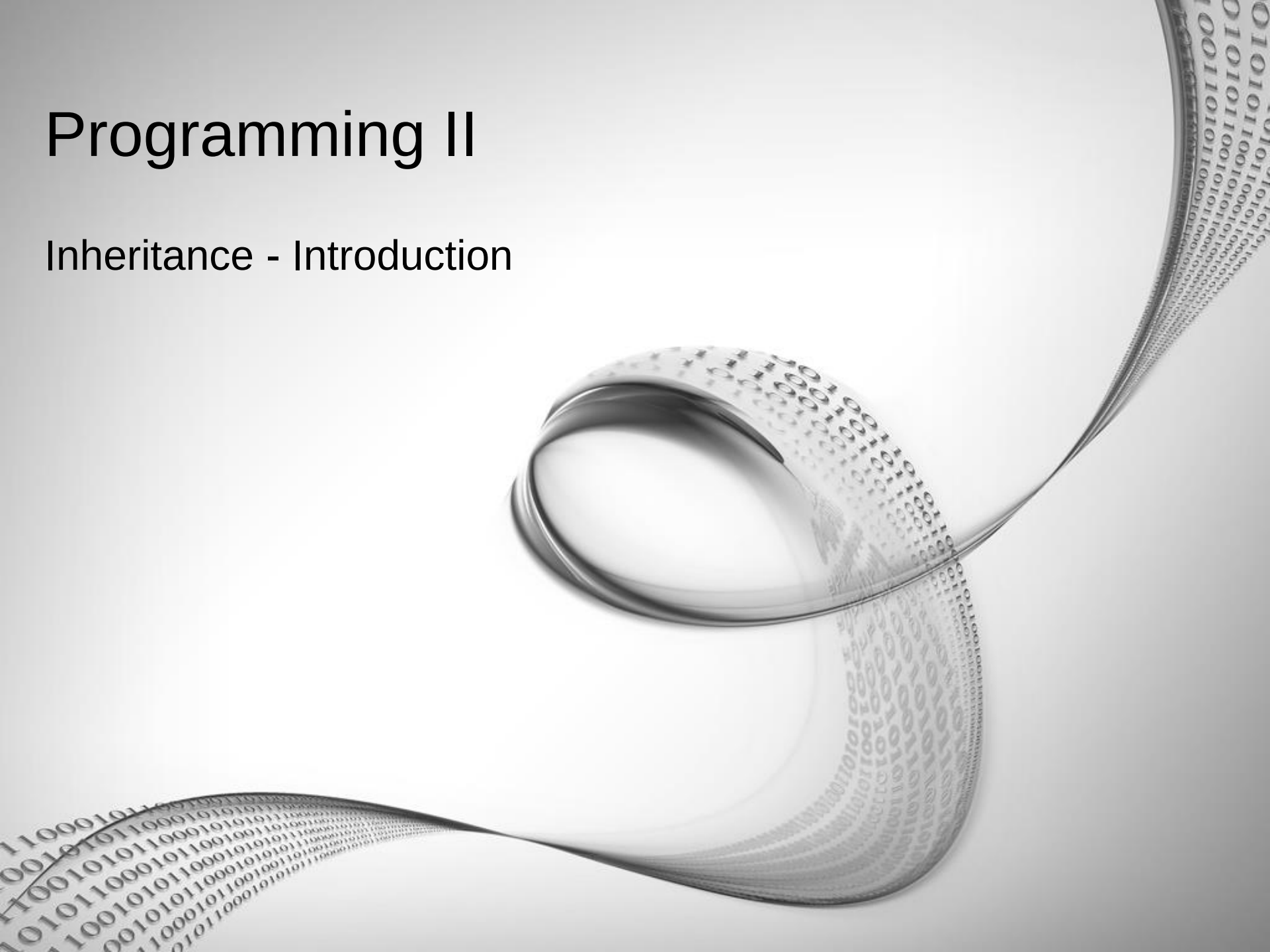
Programming II

Inheritance - Introduction



Lecture Outline

- Why we need inheritance
- Inheritance – basic principle
- Example

Why we need inheritance

What to solve?

- Reusability
  - We do not want to repeat (copy) code that we once have written and debugged.
- Extendibility
  - We want to expand (change) code that we have...

How to Use Classes?

- Reusability and extendibility in the context of the use of classes can be understood as:
  - Combining with other classes, composition
  - Extension of new behavior
  - Change the existing behavior

Composition x Inheritance

- The composition is achieved by the fact that an object of a class is composed of objects of different classes.
  - **It is a relationship "HAS".**
- Inheritance is achieved by the fact that new class is an extension or a special case of an existing class (or several classes).
  - **It is a relationship "IS".**

Example

```
class Account {
private:
    int number;
    float balance;
    float interestRate;

    Client * owner;
    Client * partner;

public:
    Account(int n, Client * o);
    Account(int n, Client * o, Client * p);
    Account(int n, Client * o, float ir);
    Account(int n, Client * o, Client * p, float ir);

    int GetCode();
    float GetBalance();
    float GetInterestRate();
    Client * GetOwner();
    Client * GetPartner();

    void Deposit(float c);
    float Withdraw(float c);
    void AddInterest();
};
```

What is wrong in the *Account* class?

- Account *with a partner* is an extension of the Account *without a partner*.
- We can use inheritance.
- How?

Parent Declaration

```
class Account {
private:
    int number;
    float balance;
    float interestRate;

    Client * owner;

public:
    Account(int n, Client * o);
    Account(int n, Client * o, float ir);

    int GetNumber();
    float GetBalance();
    float GetInterestRate();
    Client * GetOwner();

    void Deposit(float c);
    float Withdraw(float c);
    void AddInterest();
};
```

Child Declaration

```
class PartnerAccount : public Account {
private:
    Client * partner;

public:
    PartnerAccount(int n, Client * o, Client * p);
    PartnerAccount(int n, Client * o, Client * p, float ir);

    Client * GetPartner();
};
```

Implementation of Constructors

```
Account::Account(int n, Client * o){
    this->number = n;
    this->balance = 0;
    this->interestRate = (float)0.01;
    this->owner = o;
}

PartnerAccount::PartnerAccount(int n, Client * o, Client * p) : Account(n, o){
    this->partner = p;
}
```

Using (substitution)

```
int main() {
    Account * a;
    PartnerAccount *  pa;
    pa = new PartnerAccount(0, new Client(0, "hurvinek"), new Client(1, "manicka"));
    a = pa;

    cout << a->GetOwner()->GetName() << endl;
    cout << a->GetPartner()->GetName() << endl;

    cout << pa->GetPartner()->GetName() << endl;

    getchar();
    return 0;
}
```

```
class Bank {
private:
    static const int MAX_CLIENTS = 100;
    static const int MAX_ACCOUNTS = 500;

    Client * clients[MAX_CLIENTS];
    int clientsCount;

    Account * accounts[MAX_ACCOUNTS];
    int accountsCount;

public:
    Bank();
    ~Bank();

    Client * GetClient(int c);
    Account * GetAccount(int n);

    void AddClient(int c, string n);
    void AddAccount(int n, Client * o);
    void AddAccount(int n, Client * o, Client * p);
    void AddAccount(int n, Client * o, float ir);
    void AddAccount(int n, Client * o, Client * p, float ir);

    void AddInterest();
};
```

Inheritance – basic principle

Terminology

- Ancestor - descendant, the direct ancestor - descendant
- Parent – child (daughter, son)
- Super (base) class - sub class

Terminology - Examples

A

- A is base class of class B, A is parent of B, A is ancestor of C

B

- B is base class of class C, class B inherits from class A, B is parent of C

C

- C inherits from B and A, C is child of class B, C is descendant of A and direct descendant of B.

Examples

- Car - passenger car
- Tree - deciduous tree
- Collection – list, set

Correct or Incorrect?

- Car - Škoda
- Tree – pine

Generalization - specialization

- Not to be confused relationship "*is an instance*" and "*inherits from*".
  - "*Is an instance of*" is the relationship between class and object
  - "*Inherited from*" is the relationship between two classes.
  - Inheritance defines the relationship in *GENERAL- SPECIAL.*
  - Descendant should, therefore, represent a special case of an ancestor...
  - ...moreover, the ancestor should represent a generalization of its descendants.

In other words…

- Ancestor defines the common behavior of its descendants.
- Descendants behavior can be extended or changed.
- Descendants can not get rid of this behavior.
- So:
  - Descendant inherits everything without exception!!!
  - Accessibility (level of information hiding) is also inherited.

Composition and Inheritance

- Composition – „*HAS*“ x Inheritance „*IS*“
- However:
  - Inheritance can be understood as a consequence of composition.
  - Instances of descendant class contain everything of ancestor class.

Hierarchy

- When using inheritance, a hierarchy of classes is formed.
- In our case, we work with simple inheritance.
- Every child has exactly one parent. A parent may have more children.
- The simple inheritance hierarchy composes a tree.

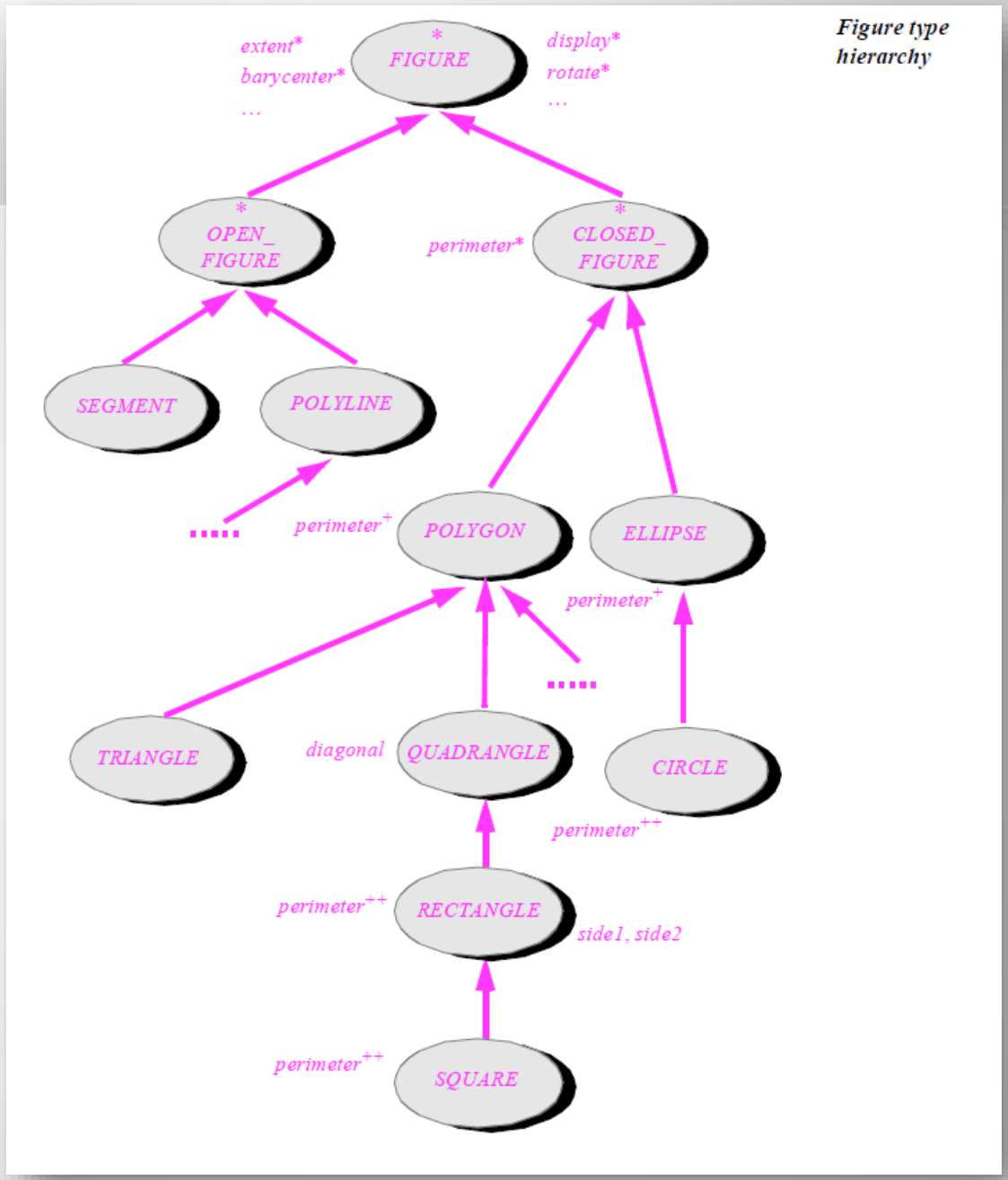


Figure type hierarchy
extent*
barycenter*
...
FIGURE
display*
rotate*
...
OPEN_ FIGURE
perimeter*
CLOSED_ FIGURE
SEGMENT
POLYLINE
perimeter+
POLYGON
ELLIPSE
perimeter+
TRIANGLE
diagonal
QUADRANGLE
CIRCLE
perimeter++
perimeter++
RECTANGLE
side1, side2
perimeter++
SQUARE

Liskov substitutional principle

- Barbara Liskov 1987. *Data abstraction and hierarchy*.
- Bertrand Meyer – *invariants of behavior*.
- An ancestor may always be represented by a descendant...
- ...and that is because they have a common behavior
- Reversely it is not true ...

Child Creation

1. Calling the object constructor.
2. Calling the ancestor constructor.
3. Execution of the ancestor constructor.
4. Execution of the object constructor.

Sources

- Bertrand Meyer. *Object-Oriented Software Construction.* Prentice Hall 1997. [459-467]

Questions

- Which two key requirements are solved by inheritance?
- For what design requirements do we use classes (what we can do with)?
- What is the difference between inheritance and composition? What do they have in common?
- What roles classes play in inheritance? Use the correct terminology.
- Explain a general relationship between a class from which is inherited and a class that inherits.
- What is inherited and what is not?
- What do we mean by simple inheritance and how it relates to the hierarchy of classes?
- What is the *Liskov Substitution Principle* and how it is applied in inheritance?
- In inheritance, what is the order of calling and execution of constructors?