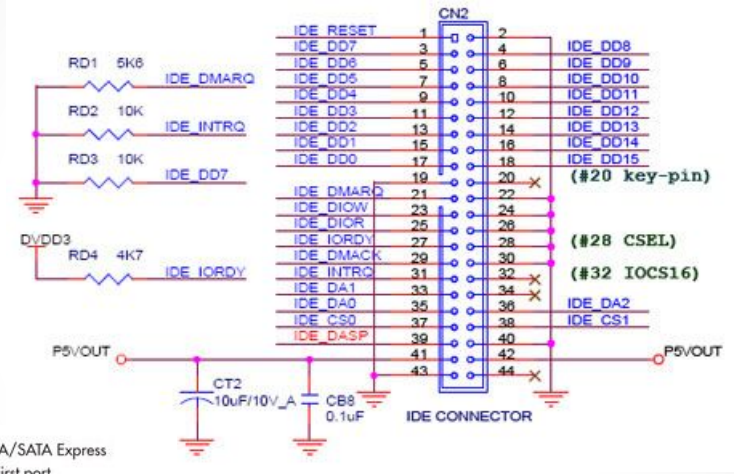
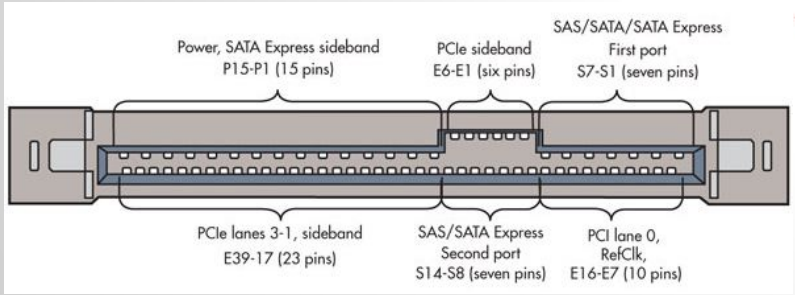
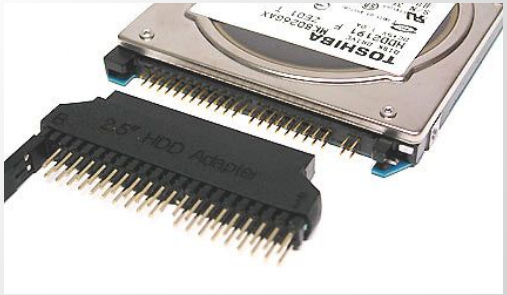
Лекция 6

Интерфейсы, наследование и полиморфизм

Интерфейс - стандарт описывающий взаимодействие с объектом







Объявление интерфейса

```
public interface IAccount {
    void PayInFunds (decimal amount);
    bool WithdrawFunds(decimal amount);
    decimal Balance();
}
```

Имплементация интерфейса

```
public class CustomerAccount : IAccount {
    private decimal m_balance = 0;
    public bool WithdrawFunds (decimal amount ){
        if (m_balance < amount)
            return false;
        m_balance -= amount;
        return true;
    }
                                    public void PayInFunds(decimal amount){
                                        m_balance += amount;
                                    }
                                    public decimal Balance() {
                                        return m_balance;
                                    }
                                }
```

Использование ссылок на интерфейс

```
public static void Main (string[] args)
{
  IAccount account = new CustomerAccount ();
  account.PayInFunds (10000);
  Console.WriteLine ("На счете {0} рублей.", account.Balance ());
}
```

```
public class ChildAccount : IAccount {
    private decimal m_balance = 0;
    public bool WithdrawFunds (decimal amount ){
        if (amount > 10 || m_balance < amount)
            return false;
        m_balance -= amount;
        return true;
    }
    public void PayInFunds(decimal amount){
        m_balance += amount;
    }
    public decimal Balance() {
        return m_balance;
    }
}
```

```
IAccount[] accounts = new IAccount[] {
    new CustomerAccount(),
    new ChildAccount(),
    new ChildAccount(),
    new CustomerAccount()
};
foreach (IAccount ac in accounts) {
    ac.PayInFunds (10000);
}
foreach (IAccount ac in accounts) {
    if (ac.WithdrawFunds (50))
        Console.WriteLine ("Сумма успешно списана со счета");
    else
        Console.WriteLine ("Не удалось списать запрошенную сумму");
}
```

Множественные интерфейсы

```
public interface IPrintable {
    string info();
}

static void print(IPrintable obj){
    Console.WriteLine(obj.info());
}

public class CustomerAccount : IAccount, IPrintable {
    ……
    public string info() {
        return string.Format("Состояние счета: {0}", m_balance);
    }
}
```

```
foreach (IAccount ac in accounts) {
    if (ac.WithdrawFunds (50))
        Console.WriteLine ("Сумма успешно списана со счета");
    else
        Console.WriteLine ("Не удалось списать запрошенную сумму");
    if (ac is IPrintable)
        print (ac as IPrintable);
}
```

```
Сумма успешно списана со счета
Состояние счета: 9950
Не удалось списать запрошенную сумму
Не удалось списать запрошенную сумму
Сумма успешно списана со счета
Состояние счета: 9950
```

Наследование классов

```
public class ChildAccount : CustomerAccount {
}

…
IAccount[] accounts = new IAccount[] {
    new CustomerAccount(),
    new ChildAccount(),
    new ChildAccount(),
    new CustomerAccount()
};
```

Переопределение родительских методов

```
public class ChildAccount : CustomerAccount {
    public override bool WithdrawFunds(decimal amount) {
        if(amount > 10 || m_balance < amount)
            return false;
        m_balance -= amount;
        return true;
    }
}
```

Уровни доступа к членам класса

- **private** - доступ разрешен только из методов класса в котором этот член объявлен
- **protected** - как **private** + методы всех дочерних классов
- **public** - доступ разрешен из любого места программы

Необходимые изменения в родительском классе

```
public class CustomerAccount : IAccount, IPrintable {
    protected decimal m_balance = 0;
    public virtual bool WithdrawFunds (decimal amount ){
.....
}
```

Полиморфизм

Возможность вызова различных реализаций методов в зависимости от того, для какого объекта был применен метод

Для создания полиморфных классов необходимо

- Объявить те методы которые должны быть переопределены с ключевым словом **virtual** в родительском классе
- В дочернем классе эти методы должны быть помечены как **override**
- Убедиться что дочерний класс имеет доступ ко всем необходимым членам родительского класса

Вызов методов родительского класса

```
public class ChildAccount : CustomerAccount {
    public override bool WithdrawFunds(decimal amount) {
        if(amount > 10)
            return false;
        return base.WithdrawFunds (amount);
    }
}
```

Замещение методов родительского класса

```
public class ChildAccount : CustomerAccount {
    public override bool WithdrawFunds(decimal amount) {
        if(amount > 10)
            return false;
        return base.WithdrawFunds (amount);
    }
    public new string info() {
      return string.Format("Состояние счета ребенка: {0}", m_balance);
    }
}
```

override VS new

- Замещенный метод не может вызывать родительский (base.WithdrawFunds)
- В случае переопределения метода, при вызове по ссылке на родительский класс будет вызван дочерний метод
- В случае замещения метода, при вызове по ссылке на родительский класс будет вызван родительский метод

Конечные классы

```
public sealed class ChildAccount : CustomerAccount {
...
}
```

Классы объявленные с ключевым словом **sealed** - не могут иметь наследников

Абстрактные классы

```
public abstract class Account : IAccount {
    private decimal m_balance = 0;
    public abstract string InsufficientFundsMessage();
    public virtual bool WithdrawFunds (decimal amount ){
        if (m_balance < amount)  return false;
        m_balance -= amount;
        return true;
    }
    public void PayInFunds(decimal amount){ m_balance += amount; }
    public decimal Balance() { return m_balance;  }
}
```

Наследование от абстрактного класса

```
public class CustomerAccount : Account {
    public override string InsufficientFundsMessage(){
        return "Недостаточно средств"; }
}
public class ChildAccount : Account {
    public override bool WithdrawFunds(decimal amount) {
        if(amount > 10)  return false;
        return base.WithdrawFunds (amount);
    }
    public override string InsufficientFundsMessage() {
        return "Сообщи родителям, что у тебя недостаточно средств";
    }
}
```

object.toString()

```
public abstract class Account : IAccount {
     private decimal m_balance ;
     private string m_name;
     public Account(string name, decimal startBalance){
          m_name = name;
          m_balance = startBalance;
     }
     public override string ToString ()
     {
        return string.Format ("Имя: {0} баланс: {1}", m_name, m_balance);
    }
…
}
```

```
public class ChildAccount : Account {
    private string m_parent;
    public ChildAccount(string name, string parent, decimal startBalance) :
        base(name, startBalance) {
        m_parent = parent;
    }
    public override string ToString ()
    {
        return base.ToString() + string.Format ("Родитель: {0}", m_parent);
    }
```

Сравнение объектов

- Нельзя просто взять и сравнить при помощи == , так как классы - это ссылочные типы
- Для сравнения объектов, а не ссылок, необходимо переопределить метод Equals()

```
public class Circle {
    private int R;
    public Circle(int r){
        R = r;
    }
    public override bool Equals (object obj)
    {
        return (obj as Circle).R == R;
    }
}
```

```
Circle a = new Circle (5);
Circle b = new Circle (5);
Circle c = a;
Console.WriteLine ("a == b: {0}\n" +
                   "a == c: {1}\n" +
                   "a eq b: {2}",
    a == b, a == c, a.Equals (b));
```

```
a == b: False
a == c: True
a eq b: True
```