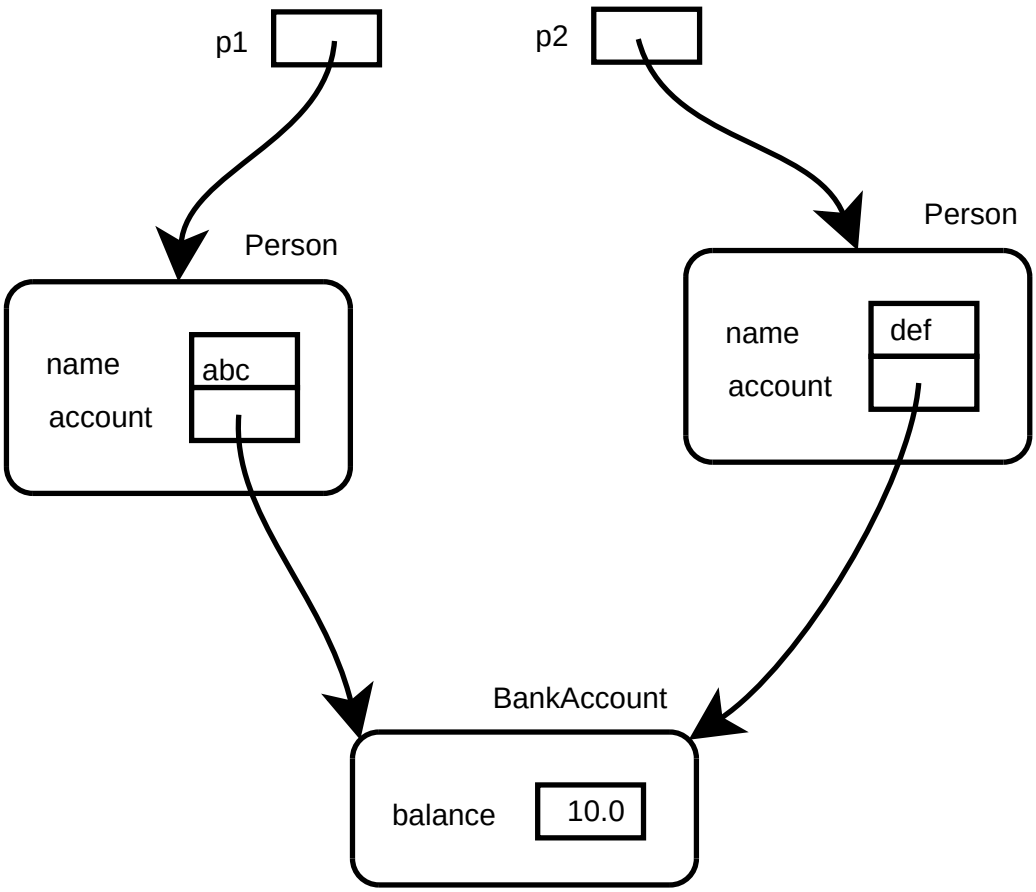
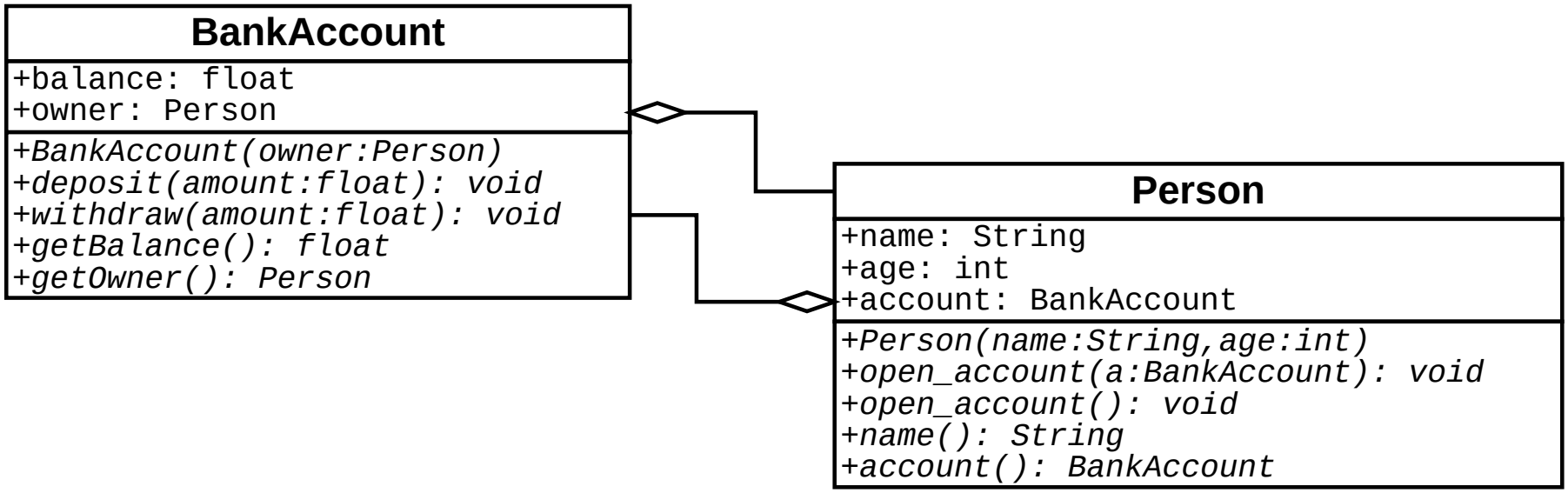
Shared references



Mutual references

- Mutual reference: An object A can have a reference to an object B which has a reference to A



Mutual reference

```
public class BankAccount
{
    private float balance;
    private Person owner;

    public BankAccount(Person owner)
    {
        this.owner = owner;
        balance = 0.0;
    }
    public void deposit(float amount)
    {
        balance = balance + amount;
    }
    public void withdraw(float amount)
    {
        if (amount <= balance)
            balance = balance - amount;
    }
    public float balance() { return balance; }
    public Person owner()  { return owner; }
}
```

Mutual reference

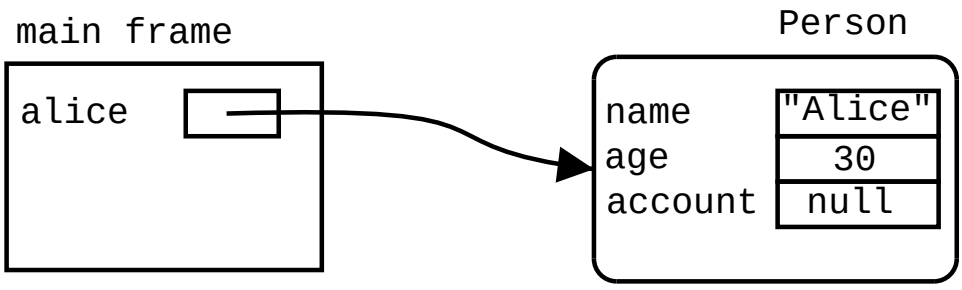
```
public class Person
{
    private String name;
    private int age;
    private BankAccount account;

    public Person(String name, int age)
    {
        this.name = name;
        this.age = age;
        account = null;
    }
    public void open_account(BankAccount a)
    {
        account = a;
    }
    public void open_account()
    {
        account = new BankAccount(this);
    }
    // Continues below...
```

```
    public String name()
    {
        return name;
    }
    public BankAccount account()
    {
        return account;
    }
}
```

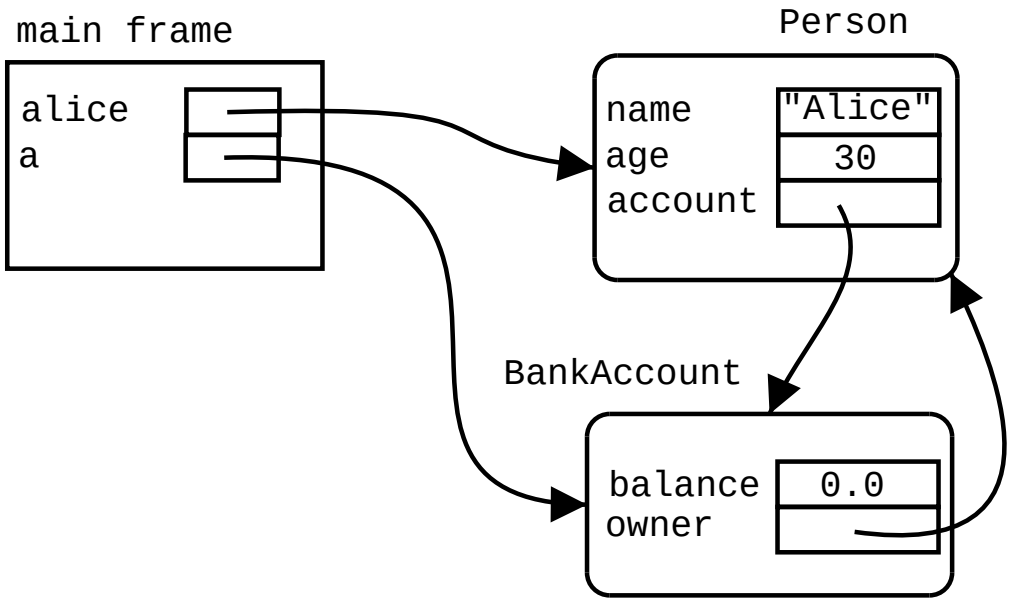
Mutual reference (contd.)

```
public class Banking
{
  public static void main(String[] args)
  {
    Person alice = new Person(“Alice”, 30);
    BankAccount a = new BankAccount(alice);
    alice.open_account(a);

    Person bob = new Person(“Bob”, 29);
    bob.open_account();

    BankAccount b = bob.account();
    b.deposit(300.0);

    alice.account().deposit(200.0f);

    System.out.println(b.balance());
    System.out.println(alice.account().balance());
    System.out.println(a.balance());
  }
}
```

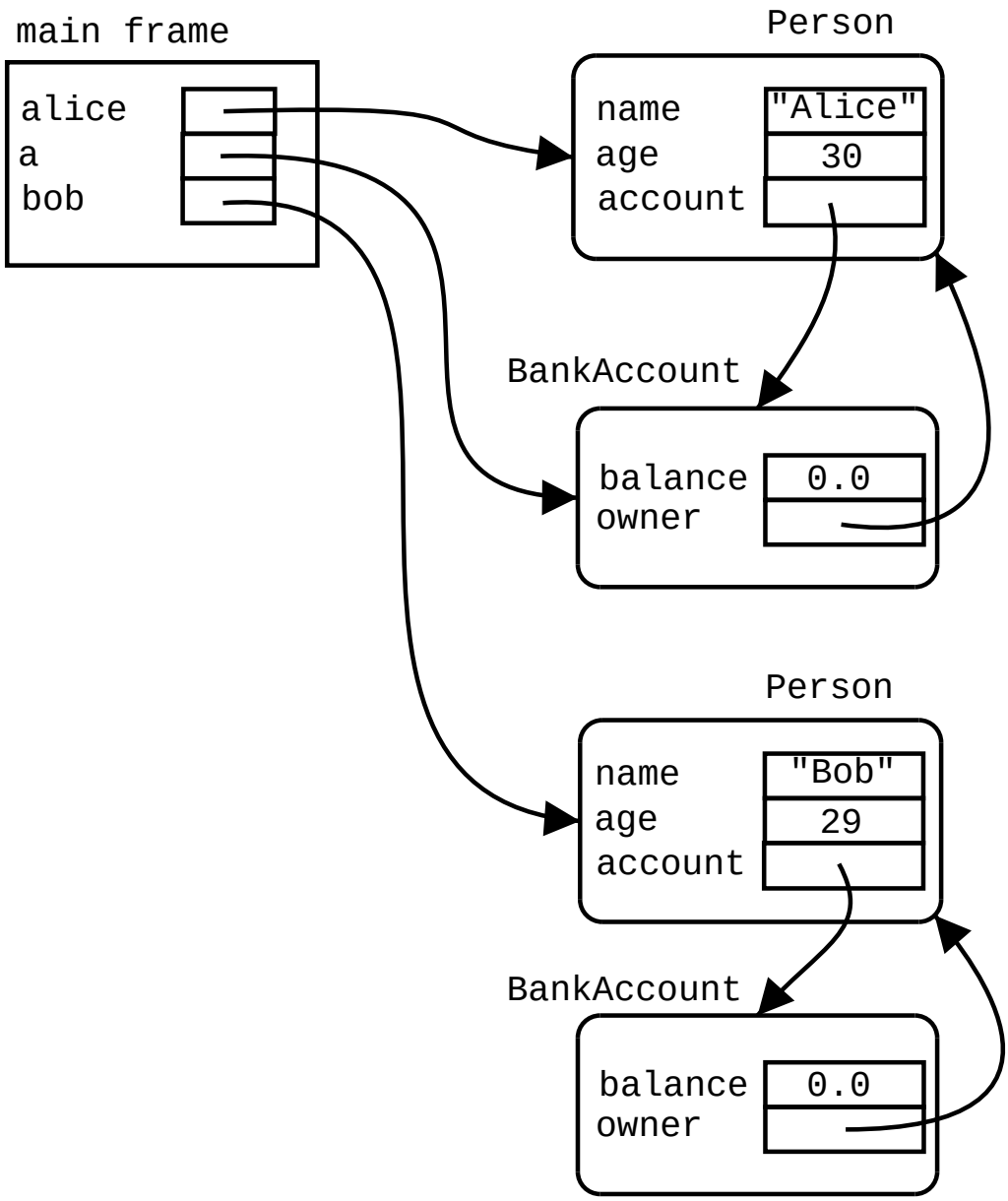
Mutual reference



Mutual reference



Mutual reference



Mutual reference

- Mutual references between objects of the same class:

```
public class Person {
    private String name;
    private Person spouse;
    public Person(String name, int age)
    {
        this.name = name;
        this.age  = age;
        this.spouse = null;
    }
    public void marry(Person someone)
    {
        this.spouse = someone;
        someone.spouse = this;
    }
    public String name()   { return name; }
    public Person spouse() { return spouse; }
}
```

Mutual reference

```
public class Marriage
{
    public static void main(String[] args)
    {
        Person a = new Person(“Alice”, 30);
        Person b = new Person(“Bob”, 29);
        a.marry(b);
        System.out.println(a.name());
        System.out.println(a.spouse().name());
        System.out.println(b.name());
        System.out.println(b.spouse().name());
    }
}
```

Method overloading

- In a given class there can be several methods with the same name...
- ...but the type or number of parameters must be different
- This is also true of constructors

Example

```
class A {
  void doSomething(int x)
  {
    System.out.println(“hello ” + x);
  }
  void doSomething(boolean x)
  {
    System.out.println(“good bye ” + x);
  }
}
```

Example

```
class B {
  void p()
  {
    A x;
    x = new A();
    x.doSomething(3);
    x.doSomething(false);
  }
}
```

Example

```
class B {
  void p()
  {
    A x;
    x = new A();
    x.doSomething(3);
    x.doSomething(“false”);
  }
}
```

Example

```
class A {
  void doSomething(int x)
  {
    System.out.println(“hello ” + x);
  }
  void doSomething(int x, int y)
  {
    System.out.println(“good bye ” + (x+y));
  }
}
```

Example

```
class B {
  void p()
  {
    A x;
    x = new A();
    x.doSomething(3);
    x.doSomething(4,5);
  }
}
```

Example

```
class C
{
  void f()
  {
    System.out.println(1);
  }
}
class D
{
  void g()
  {
    System.out.println(2);
  }
}
```

Example

```
class A {
  void doSomething(C x)
  {
    System.out.println(“hello ” + x.f());
  }
  void doSomething(D x)
  {
    System.out.println(“good bye ” + x.g());
  }
}
```

Example

```
class B {
  void p()
  {
    A x = new A();
    C y = new C();
    D z = new D();
    x.doSomething(y);
    x.doSomething(z);
  }
}
```

Method overloading

- In a given class there can be several methods with the same name...
- ...but the type or number of parameters must be different
- This is also true of constructors

Example

```
public class CheckingAccount {
  double balance;
  CheckingAccount() { balance = 0.0; }
  void deposit(double amount)
  {
    balance = balance + amount;
  }
}
```

Example

```
public class MagicAccount {
  double balance;
  MagicAccount() { balance = 0.0; }
  void deposit(double amount)
  {
    balance = balance + amount + 100.0;
  }
}
```

Example

```
public class BankingApplication {
  public static void main(String[] args)
  {
    CheckingAccount a = new CheckingAccount();
    MagicAccount b = new MagicAccount();
    a.deposit(500.0);
    b.deposit(300.0);
  }
}
```

Example

```
public class CanadianDollars {
  double amount, rate;

  CanadianDollars(double a)
  {
    amount = a;
    rate = 0.75;
  }

  double USvalue()
  {
    return amount * rate;
  }
}
```

Example

```
public class Euros {
  double amount, rate;

  Euros(double a)
  {
    amount = a;
    rate = 1.24;
  }

  double USvalue()
  {
    return amount * rate;
  }
}
```

Example

```
public class MagicAccount {
  double balance;

  MagicAccount() { balance = 0.0; }

  void deposit(double amount)
  {
    balance = balance + amount + 100.0;
  }
}
```

Example

```
public class MagicAccount {
  double balance;

  MagicAccount() { balance = 0.0; }

  void deposit(CanadianDollars amount)
  {
    balance = balance
            + amount.USvalue() + 200.0;
  }
  void deposit(Euros amount)
  {
    balance = balance
            + amount.USvalue() + 100.0;
  }
}
```

Example

```
public class BankingApplication {
  public static void main(String[] args)
  {
    CheckingAccount a = new CheckingAccount();
    MagicAccount b = new MagicAccount();
    a.deposit(500.0);

    CanadianDollars dollars;
    Euros eus;
    dollars = new CanadianDollars(300.0);
    eus = new Euros(100.0);
    b.deposit(dollars);
    b.deposit(eus);
  }
}
```

Static variables and methods

- Declaring an instance variable

  type identifier ;

- Declaring visibility of the attribute

  modifier type identifier ;

  where *modifier* is `public`, `private` or `protected`

- Declaring an attribute as static (only for attributes, not local variables)

  modifier `static` *type identifier* ;

Declaring methods

- Declaring normal methods

```
type method_name(type1 arg1, type2 arg2,
                 ..., typen argn)
{
    statements;
}
```

- Declaring static methods

```
static type method_name(type1 arg1, type2 arg2,
                        ..., typen argn)
{
    statements;
}
```

Static variables

- The attributes of a class are normal variables.
- The values of these attributes are individual to each object in a class.

```
public class A {
  int x;
}
public class B {
  void m()
  {
    A u = new A();
    A v = new A();
    u.x = 5;
    v.x = -7;
    // Here, u.x == 5 and v.x == -7
  }
}
```

Static variables (contd.)

- Static variables are attributes of the class, not of the objects
- Static variables are shared between all the objects in a class

```
public class A {
  static int x;
}
public class B {
  void m()
  {
    A u = new A();
    A v = new A();
    u.x = 5;
    v.x = -7;
    // Here, u.x == -7 and v.x == -7
  }
}
```

Static variables (contd.)

```
public class BankAccount
{
  float balance;

  BankAccount()
  {
    balance = 0.0f;
  }
  void deposit(float amount)
  {
    balance = balance + amount;
  }
  void withdraw(float amount)
  {
    if (amount < balance)
      balance = balance - amount;
  }
}
```

Static variables (contd.)

```
public class Bank {
  public static void main(String[] args)
  {
    BankAccount pete, amy;
    pete = new BankAccount();
    amy = new BankAccount();

    pete.deposit(700.0f);
    amy.deposit(800.0f);

    System.out.println(pete.balance);
    System.out.println(amy.balance);
  }
}
```

Static variables (contd.)

```
public class BankAccount
{
  static float balance;

  BankAccount()
  {
    balance = 0.0f;
  }
  void deposit(float amount)
  {
    balance = balance + amount;
  }
  void withdraw(float amount)
  {
    if (amount < balance)
      balance = balance - amount;
  }
}
```

Static variables (contd.)

```
public class Bank {
  public static void main(String[] args)
  {
    BankAccount pete, amy;
    pete = new BankAccount();
    amy = new BankAccount();

    pete.deposit(700.0f);
    amy.deposit(800.0f);

    System.out.println(pete.balance);
    System.out.println(amy.balance);
  }
}
```

Static methods

- Normal (non-static) methods represent the behaviour of objects
- Static methods are not associated with objects
- Static methods are only "services" provided by a class
- For example:
  - Keyboard.readString
  - Keyboard.readInt
  - Math.sqrt
  - Math.pow
  - ...etc

Calling normal methods

- When calling a non-static method, the syntax is

  objectreference.*method_name*(*arg1*,*arg2*,...,*argn*)

  where variable has a reference to an object (e.g. *objectreference* = new *MyClass*();)

  For example:

```
String title = new String("Lock, Stock");
int size = title.length();
char initial = title.charAt(0);
```

Calling static methods

- When calling a static method, the syntax is

  class_name.*method_name*(*arg1*,*arg2*,...,*argn*)

  Forexample:

```
double power = Math.pow(2.0, 3);
int n = Keyboard.readInt();
```

Example

```
public class A
{
    void p()
    {
        System.out.println(“Hello”);
    }
    static void q()
    {
        System.out.println(“Good bye”);
    }
}
```

(Note: Classes can have both static and non-static methods)

Calling static methods

- A call to a static method takes the form

  class_name.*method*(*arg1*,*arg2*,...,*argn*)

- When the method is called, the corresponding frame does not have a reference to `this`, because there is no object receiving the message.

- It can also take the form

  object_reference.*method*(*arg1*,*arg2*,...,*argn*)

- But the object will be ignored

Example (contd.)

```
public class B
{
    public static void main(String[] args)
    {
        A.q();          // Prints Good bye
        A x = new A();  // Creates an A object
        x.p();          // Prints Hello
        A.p();          // Compile-time Error
        x.q();          // Prints Good bye
    }
}
```

Static methods access

- Since the frame of a static method does not have a reference to an object, static methods cannot access attributes of an object

```
public class A
{
  int n;
  void p()
  {
    System.out.println(n); //OK
  }
  static void q()
  {
    System.out.println(n); //WRONG
  }
}
```

Static methods access

- Since the frame of a static method does not have a reference to an object, static methods cannot access attributes of an object

```
public class A {
  int n;
  void p()
  {
    System.out.println(this.n); //OK
  }
  static void q()
  {
    System.out.println(this.n); //WRONG
  }
}
```

Static methods access

- A static method can be called from a non-static context, but...
- A non-static method cannot be called from a static context, because in order to call a non-static method, you need to provide a reference to an object.

Accessing static methods from non-static methods

```
public class A
{
    void p()
    {
        System.out.println(“Hello”);
        q();
    }
    static void q()
    {
        System.out.println(“Good bye”);
    }
}
```

... is OK

Accessing static methods from non-static methods

```
public class A
{
    void p()
    {
        System.out.println(“Hello”);
        this.q();
    }
    static void q()
    {
        System.out.println(“Good bye”);
    }
}
```

Accessing static methods from non-static methods

```
public class A
{
    void p()
    {
        System.out.println(“Hello”);
        A.q();
    }
    static void q()
    {
        System.out.println(“Good bye”);
    }
}
```

Accessing non-static methods from static methods

```
public class A
{
    void p()
    {
        System.out.println("Hello");
    }
    static void q()
    {
        System.out.println("Good bye");
        p();
    }
}
```

... is **not** OK, because in method q, there is no reference "`this`" to an object to which the message "`p()`" would be sent.

Accessing non-static methods from static methods

```
public class A
{
    void p()
    {
        System.out.println(“Hello”);
    }
    static void q()
    {
        System.out.println(“Good bye”);
        this.p();
    }
}
```

When to use each kind of method

- Non-static methods are used to describe the behaviour of objects.
- Static methods are used to describe functions, or services that a class provides, independently of any object of that class.

The end