Objects are “first class citizens”

- Since classes are data types and objects are their values, then we can do with objects the “same” things that we can do with primitive values, namely:
  - We can assign objects to variables,
  - We can pass objects as arguments to methods, and
  - Methods can return objects as their result.

Objects are "first class citizens" (contd.)

- Variables, attributes can be declared as having a class for its type:

```
Stereo mystereo, yourstereo;
```

- Variables whose type is a class can be assigned objects of that class:

```
mystereo = new Stereo();
yourstereo = mystereo;
```

- Objects can be passed as parameters; if there is a method `void m(Stereo s) {...}` in some class `C`, then we can do:

```
C x = new C();
x.m(mystereo);
x.m(yourstereo);
x.m(new Stereo());
```

Objects are "first class citizens" (contd.)

- Objects can be returned as values; if there is a method

```
Stereo p()
{
    return new Stereo();
}
```

in some class C, then we can do:

```
C x = new C();
mystereo = x.p();
```

...provided that the variable which is being assigned is of the same type.

Objects as first class values

- Objects can be attibutes of other objects

```
public class Rabbit {
  void jump() { ... }
}
public class Cage {
  Rabbit my_rabbit;
  void put(Rabbit a)
  {
    my_rabbit = a;
  }
  Rabbit get()
  {
    return my_rabbit;
  }
}
```

...elsewhere...

```
Rabbit bugs = new Rabbit();
Cage c = new Cage();
c.put(bugs);
Rabbit wester = c.get();
```

Classes are data types

```
public class Theater {
  void play(Movie m)
  {
    m.print();
  }
}

public class MovieApplication3 {
  public static void main(String[] args)
  {
    Movie m1;
    Theater t = new Theater();
    m1 = new Movie(“Les Invasions barbares”,
                   “Denys Arcand”);
    t.play(m1);
  }
}
```

Example

```
public class A {
    int k;
    A()        // Constructor
    {
        k = 1;
    }
}
public class B {
    A x;         // Objects can be attributes;
    void m()
    {
        x = new A();
    }
    void p(A u) // Parameters may have a class
    {           //    for type
        x = u;  // The object u is created
    }           //    elsewhere
    A r()
    {
        return x;
    }
}
```

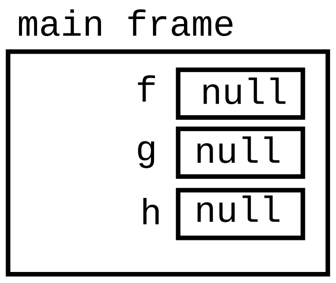
McGill

Example (contd.)

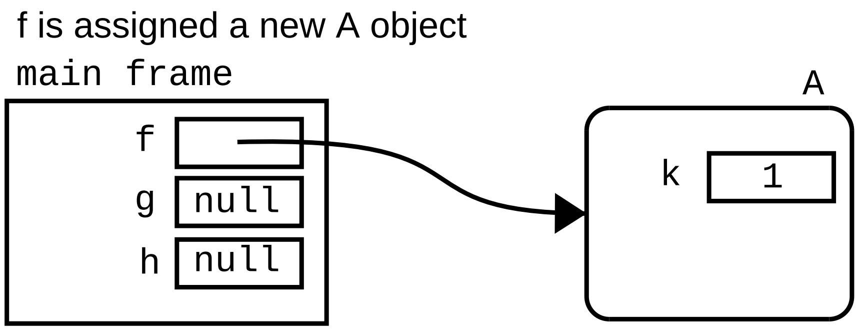
```
public class C {
  public static void main(String[] args)
  {
    A f,g;  // f and g are initialized to null
    B h;    // h is initialized to null
    f = new A();
    // Here f.k is 1
    f.k = 5;
    h = new B();
    h.m();    // assigns a new A to h.x, so ...
    // Here h.x.k is 1
    h.p(f);   // object f is passed as argument
    // Here h.x is f, and therefore h.x.k is 5
    // Also, g is still null, so there is no g.k
    g = h.r();
    // Now g is the same as h.x, which is f,
    // ...so g.k is 5
  }
}
```

Example (contd.)

The variables are initialized to null

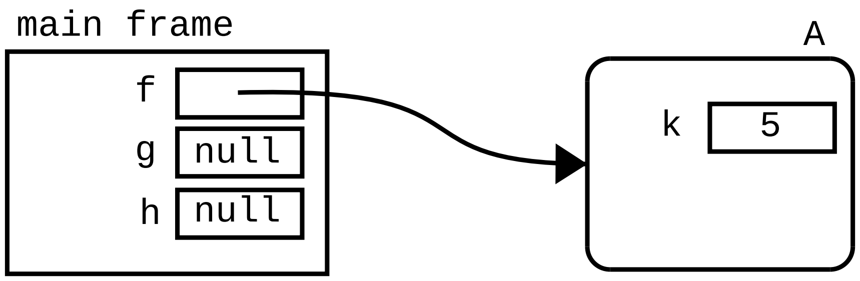


Example (contd.)



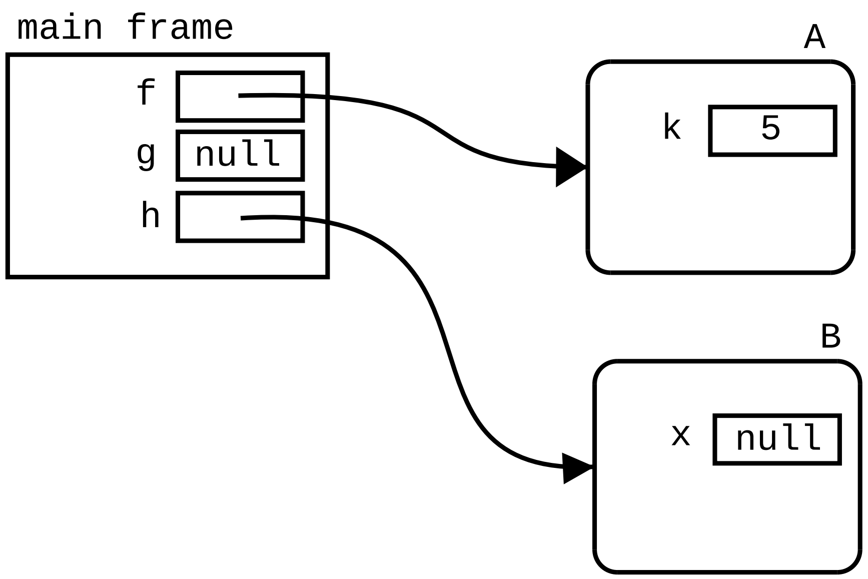
Example (contd.)

The statement `f.k = 5;` is executed



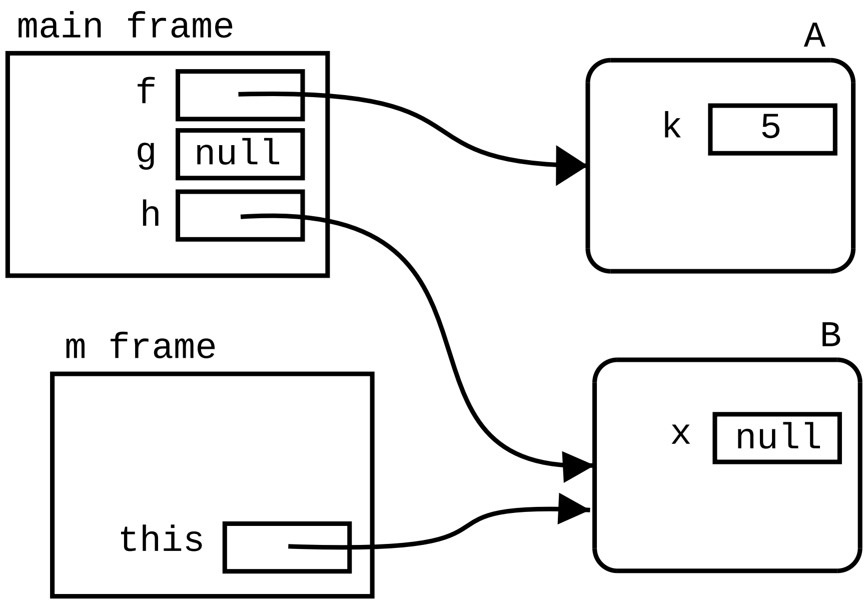
Example (contd.)

h is assigned a new B



Example (contd.)

We call `h.m()` which creates a frame for m with no arguments

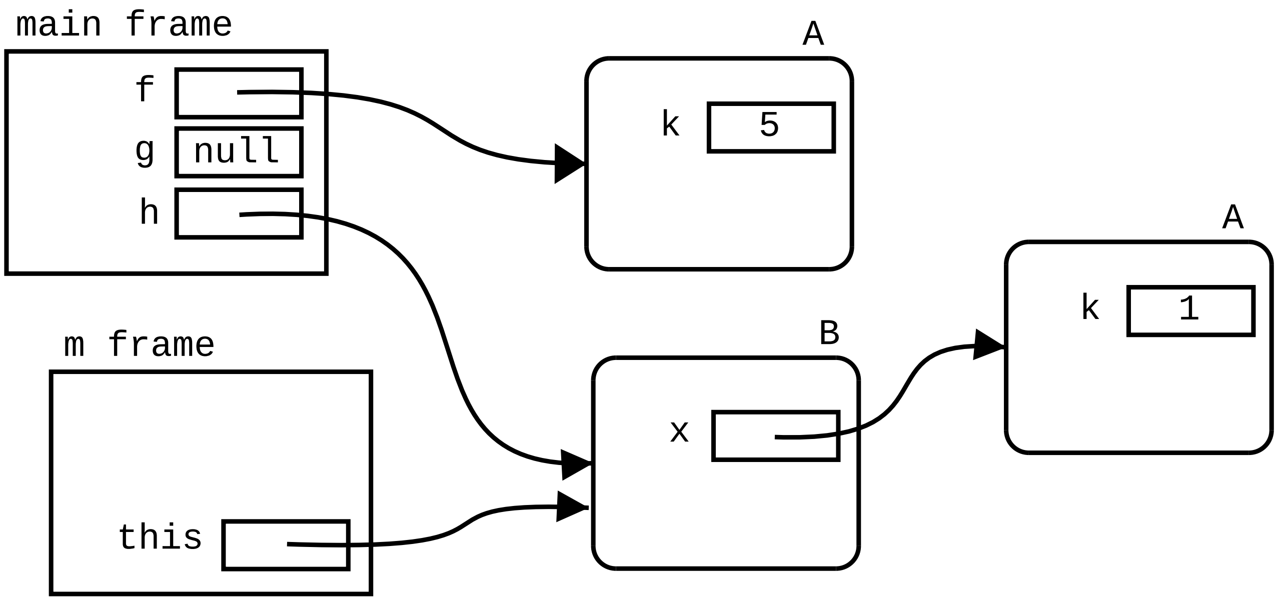


Example (contd.)

The body of m is executed. It consists of the single statement

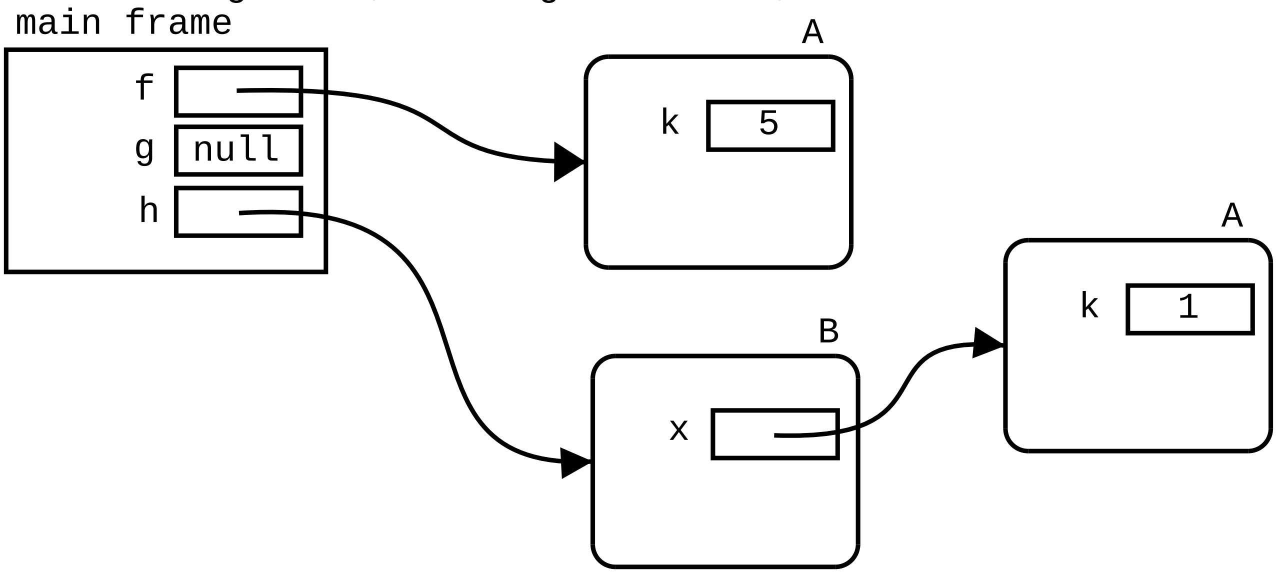
```
x = new A();
```

which creates a new A object and assigns it to `this.x`



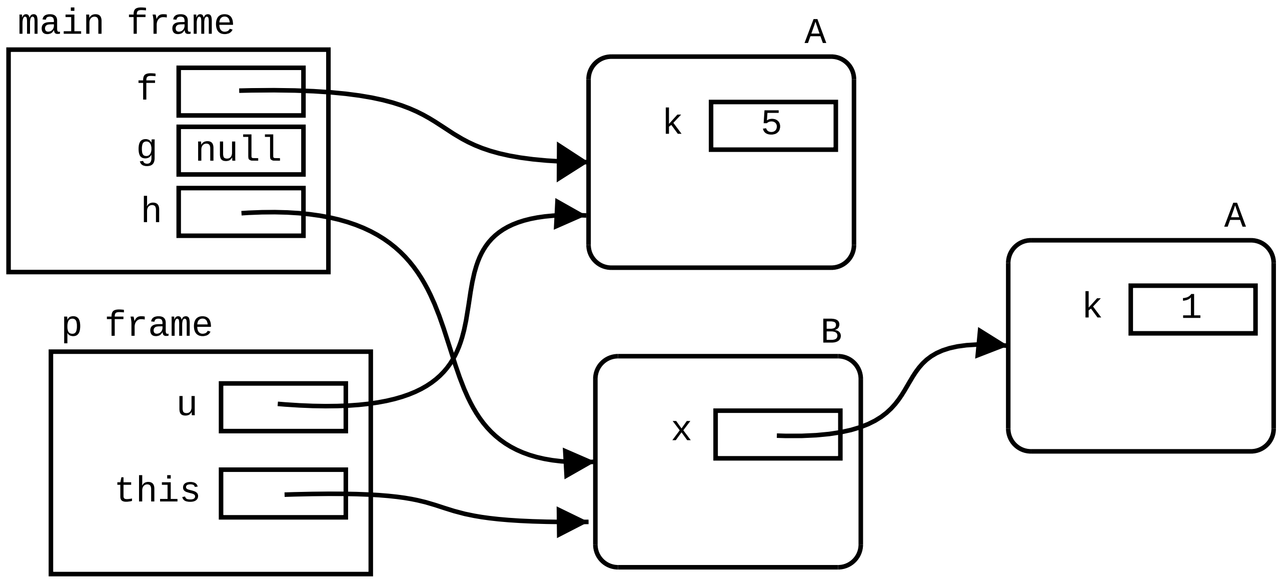
Example (contd.)

After returning from m, its frame gets discarded, and `h.x.k` is `1`



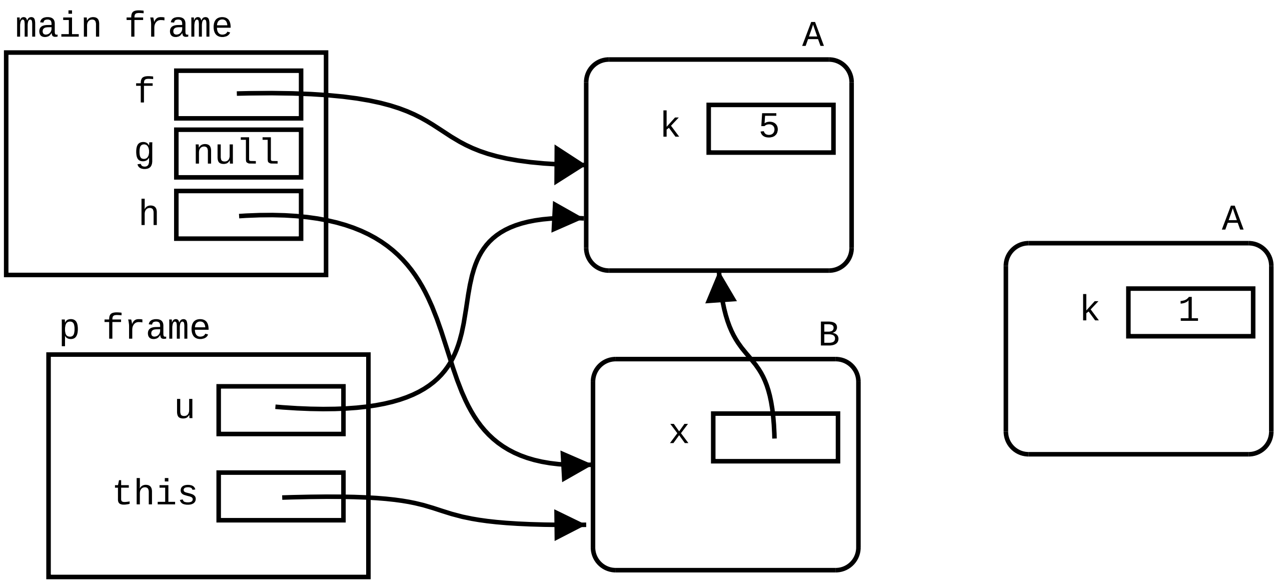
Example (contd.)

Computation in main continues with `h.p(f);`
A frame for p is created, assigning f to its parameter u



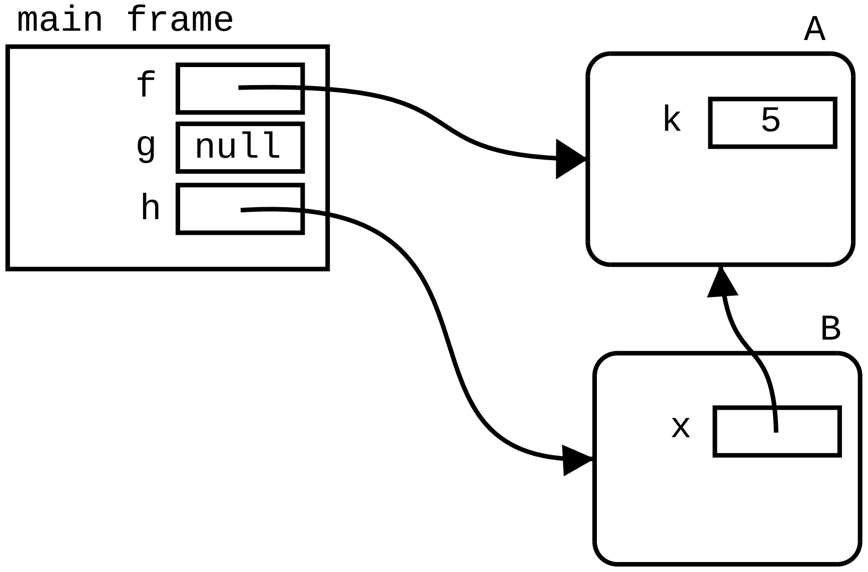
Example (contd.)

In this frame, the body of p is executed.
The body of p is `x = u;` which is the same as `this.x = u;`



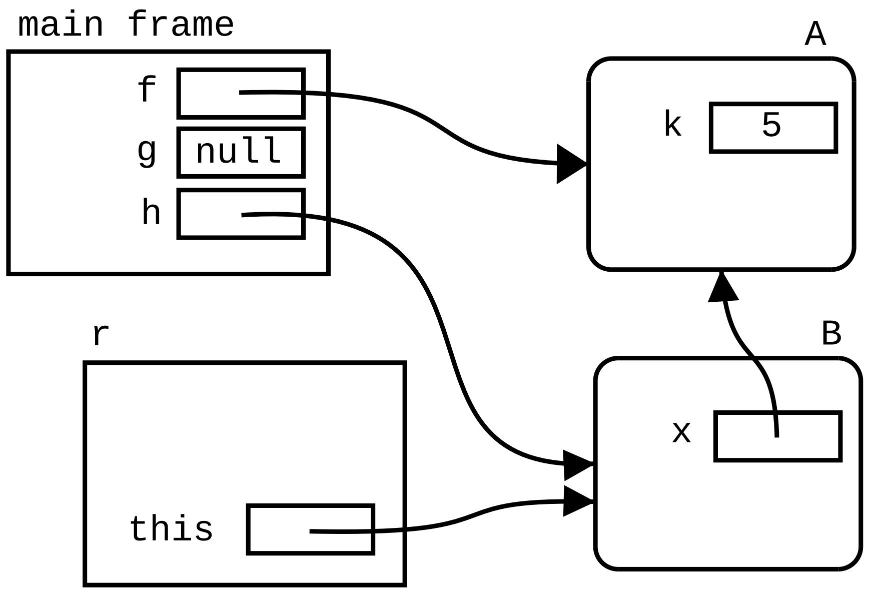
Example (contd.)

When p ends, its frame is discarded.
The other A object that has no references to it, is also discarded.



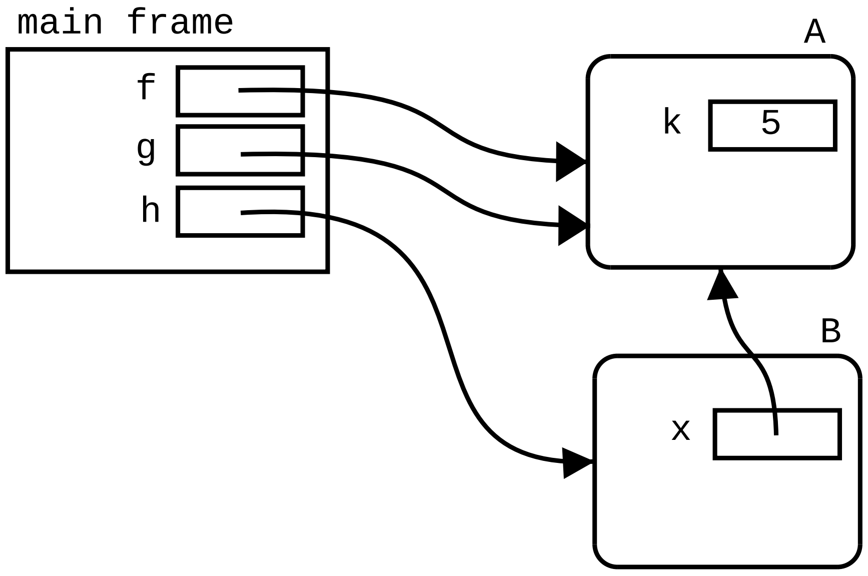
Example (contd.)

Computation is resumed with the next instruction of the main: `g = h.r();` So the right-hand side of the assignment, `h.r()` is evaluated. So a frame for r is created.



Example (contd.)

The body of r is executed in this frame. Its body is `return x;` which is the same as `return this.x;` But `this.x` is the same as the pointer to f, so this pointer is returned, discarding the frame for r, and performing the pending assignment to g, which is now equivalent to: `g = h.x;` or `g = f;`



Encapsulation and visibility

- Abstraction and visibility
- Purpose of encapsulation:
  - Hiding the state of an object, (part of) the structure of an object (attributes and/or methods,) or the internal representation of data, so that a client doesn't have to know about the internals of an object (abstraction.)
  - Security: maintaining the integrity of data. Enforcing limited visibility so that clients cannot "corrupt" the state of an object, so that only the class of the object can change the object's state.
- Visibility modifiers (for attributes and methods): `public`, `private` and `protected`.
- Visibility modifiers are orthogonal (independent) of whether the attribute or method is static or not. So they can be combined in any way.

Encapsulation to enforce integrity

```
public class BankAccount
{
  public double balance;
  public BankAccount() { balance = 0.0; }
  public void deposit(double amount)
  {
    if (amount > 0.0)
      balance = balance + amount;
  }
  public void widthdraw(double amount)
  {
    if (amount <= balance)
      balance = balance - amount;
  }
}
```

Encapsulation to enforce integrity

```
public class BankingApplication
{
  public static void main(String[] args)
  {
    BankAccount b;
    b = new BankAccount();
    b.deposit(500.0);
    b.balance = b.balance - 700.0;   // OK
  }
}
```

Encapsulation to enforce integrity

```
public class BankAccount
{
  private double balance;
  public BankAccount() { balance = 0.0; }
  public void deposit(double amount)
  {
    if (amount > 0.0)
      balance = balance + amount;
  }
  public void widthdraw(double amount)
  {
    if (amount <= balance)
      balance = balance - amount;
  }
}
```

Encapsulation to enforce integrity

```
public class BankingApplication
{
  public static void main(String[] args)
  {
    BankAccount b;
    b = new BankAccount();
    b.deposit(500.0);
    b.balance = b.balance - 700.0;   // ERROR
  }
}
```

Encapsulation to enforce integrity

```
public class BankingApplication
{
  public static void main(String[] args)
  {
    BankAccount b;
    b = new BankAccount();
    b.deposit(500.0);
    b.withdraw(700.0);   // OK
  }
}
```

Privacy is relative

```
public class BankAccount
{
  private double balance;
  public BankAccount() { balance = 0.0; }
  public void deposit(double amount)
  {
    if (amount > 0.0)
      balance = balance + amount;
  }
  public void widthdraw(double amount)
  {
    if (amount <= balance)
      balance = balance - amount;
  }
  public void transfer(BankAccount other)
  {
    this.balance = this.balance + other.balance;
    other.balance = 0.0;
  }
}
```

Privacy is relative

```
public class BankingApplication
{
  public static void main(String[] args)
  {
    BankAccount b1, b2;
    b1 = new BankAccount();
    b2 = new BankAccount();
    b1.deposit(500.0);
    b2.transfer(b1);
  }
}
```

McGill

The end