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();
```

Encapsulation and visibility

- Abstraction and visibility
- Hiding the state of an object
- Hiding (part of) the structure of an object (attributes and/or methods.)
- Hiding from clients
- 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.

Visibility modifiers for attributes

- A normal attribute has the syntax:

  ```
  type variable;
  ```

- With a modifier:

  ```
  modifier type variable;
  ```

- So there are three forms:

  ```
  public type variable;
  private type variable;
  protected type variable;
  ```

- The default is `protected`.

- In general:

  ```
  [modifier] type variable [= expression];
  ```

McGill

Visibility modifiers for methods

- A normal method has the syntax:

```
type method(type1 param1, type2 param2,
            ..., typen paramn)
{
    statements;
}
```

- In general:

```
[modifier] type method(type1 param1,
                       type2 param2,
                       ...,
                       typen paramn)
{
    statements;
}
```

Encapsulation

- A `public` variable or method can be accessed by any method in any class anywhere at any time
- A `private` variable or method can be accessed only by objects of the same class, this is, only by methods of the same class.
- ...but a `private` variable or method can be accessed by all objects of the same class
- A `protected` variable or method can be accessed by any method in any class *in the same package only, or by subclasses*

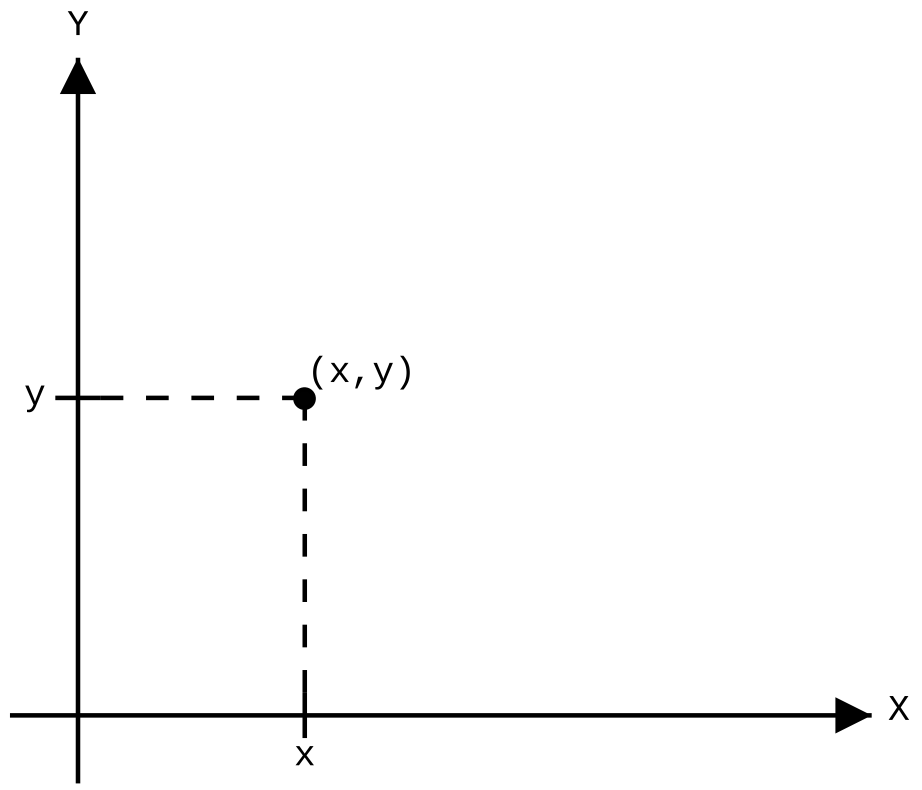
Hiding implementation

- Points have alternative characterizations:
  - Rectangular coordinates
  - Polar coordinates

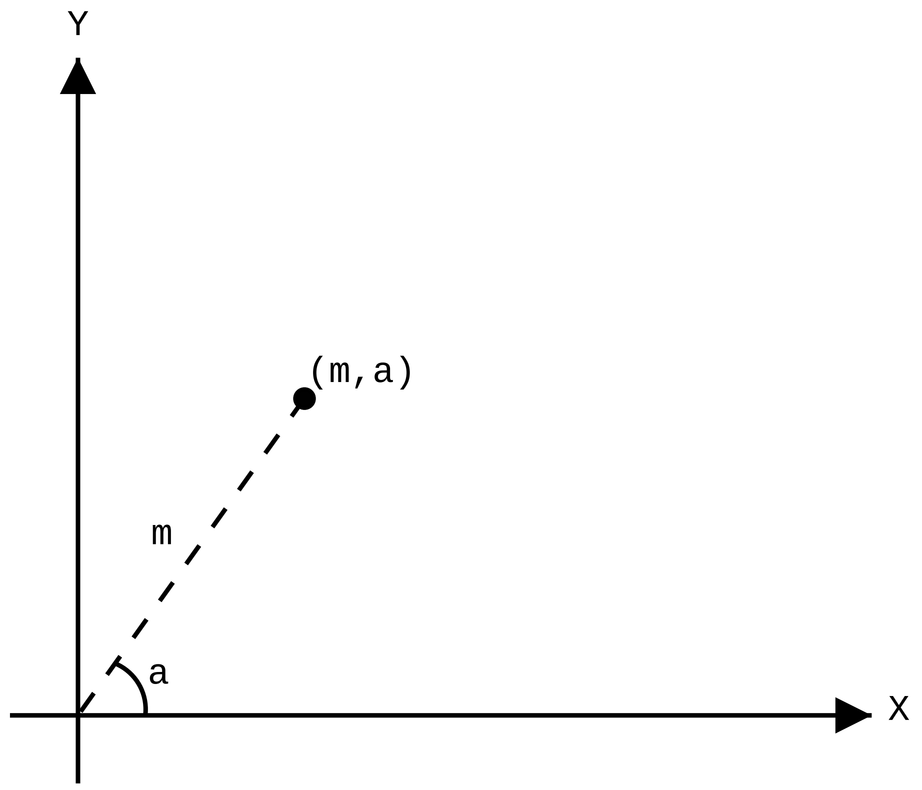
- If a point is represented using rectangular coordinates, you would like to be able to know what are its polar coordinates and viceversa.

- We should implement a class in a way which makes its clients independent of the internals of the class. This is, we can implement a class in any way as long as its clients do not need to be changed. (Abstraction implies Modularity and Integration.)

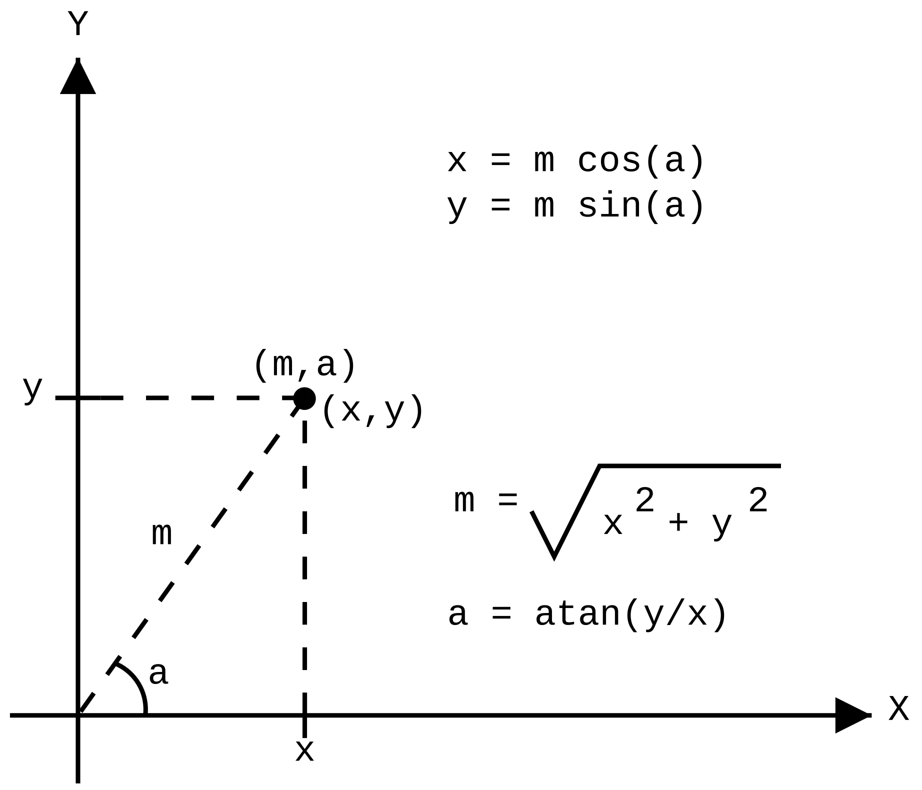
Point representation



Point representation



Point representation



$x = m \cos(a)$
$y = m \sin(a)$

$m = \sqrt{x^2 + y^2}$

$a = \operatorname{atan}(y/x)$

Point operations

- Accessor methods:
  - get_x
  - get_y
  - get_angle
  - get_magnitude
- Mutator methods
  - set_xy
  - set_angle_and_magnitude

Underlying rectangular representation

```
public class Point
{
  private double x, y;

  public void set_xy(double x, double y)
  {
    this.x = x;
    this.y = y;
  }
  public double get_x()
  {
    return x;
  }
  public double get_y()
  {
    return y;
  }

  // Continues below ...
  public double get_angle()
  {
    return Math.atan2(y, x);
  }
  public double get_magnitude()
  {
    return Math.sqrt(x*x + y*y);
  }
  public void set_angle_and_magnitude(double a,
                                      double r)
  {
    x = r * Math.cos(a);
    y = r * Math.sin(a);
  }
}
```

Underlying polar representation

```
public class Point
{
    private double angle, magnitude;
    public double get_angle()
    {
        return angle;
    }
    public double get_magnitude()
    {
        return magnitude;
    }
    public void set_angle_and_magnitude(double a,
                                        double r)
    {
        angle = a;
        magnitude = r;
    }

    // Continues below ...
    public void set_xy(double x, double y)
    {
        magnitude = Math.sqrt(x*x + y*y);
        angle = Math.atan2(y, x);
    }
    public double get_x()
    {
        return magnitude * Math.cos(angle);
    }
    public double get_y()
    {
        return magnitude * Math.sin(angle);
    }
}
```

Aliases and shared references

- Variables and values
- If we execute:

```
    x = 5;
```

- then the value of x is 5.
- Strictly speaking x is not 5; x is a memory location.
- So while we would informally read x==5 as "x is 5", the actual meanning is the value of x is 5.
- Hence, after executing

```
    x = 5;
    y = 5;
```

  and both x and y have the same value, but they are not the same variable.

Variables and values

- For primitive data types (int, boolean, float, String, etc.)

  ```
  x = y;
  ```

  means copy the value of y in the memory location of x;

- So

  ```
  int x, y;
  x = 4;
  y = x;
  ```

  means that both `x` and `y` have value 4, but they have a separate identity because each of them is a different memory location...

x	4
y	4

Variables and values

- So the value of y is the same as the value of x, but y is not the same as x.
- ... which implies that their values are independent:

```
int x, y;
x = 4;
y = x;
x++;
// x == 5 and y == 4
```

- Variables can be changed over time by assignment.
- If x and y are two variables of a primitive data type, we say that they are equal if their values are the same.
- We can test for whether the values of two variables are the same using the == operator.

Being the “same” as something else

- Suppose we have

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

- Both variables `x` and `y` are `A`'s
- ... but the objects they refer to are different, individual, and independent `A`'s.

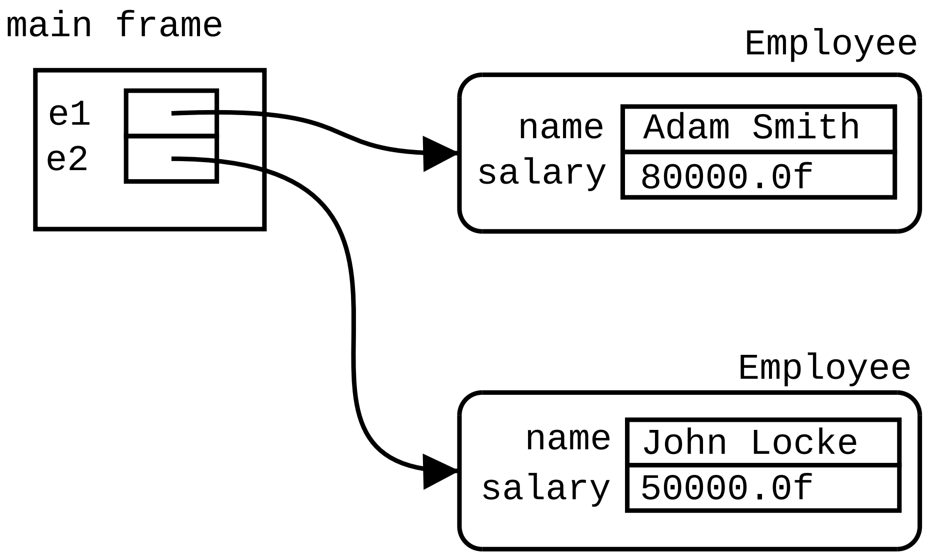
Example:

```
class Employee
{
    String name;
    float salary;
    Employee(String name, float salary)
    {
        this.name = name;
        this.salary = salary;
    }
    String name() { return name; }
    float salary() { return salary; }
    void raise_salary(float percentage)
    {
        salary = salary * (1 + percentage/100.0f);
    }
}
```

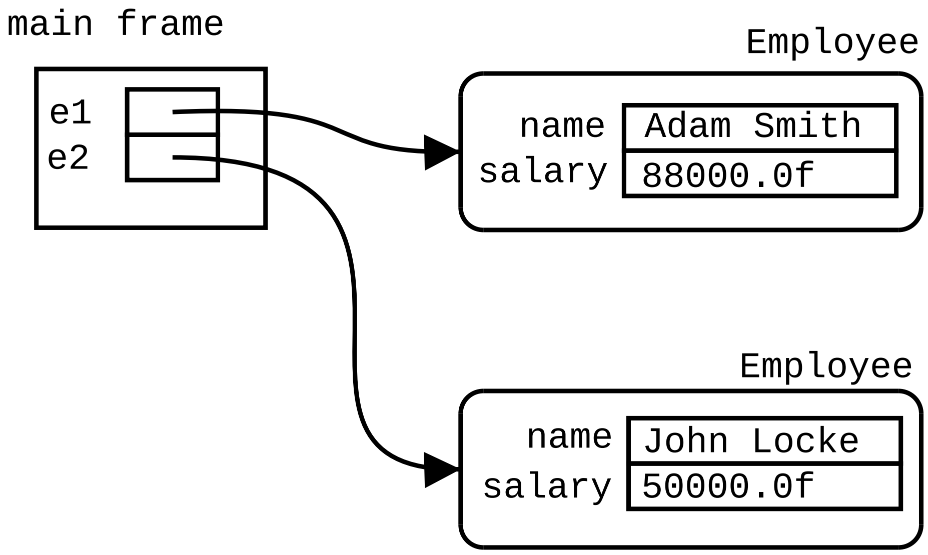
Example (contd.)

```
public class Test
{
  public static void main(String[] args)
  {
    Employee e1 = new Employee(“Adam Smith”, 80000
    Employee e2 = new Employee(“John Locke”, 50000
    e1.raise_salary(10f);
    System.out.println(e2.salary());
  }
}
```

Example (contd.)



Example (contd.)



Alias

- A variable is an alias of another variable if they both point to the same object.

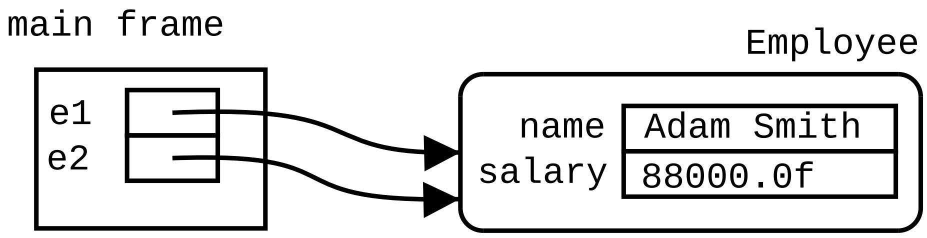
```
A x, y;
x = new A();
y = x;
```

- In this case `x` and `y` are the "same".
- More precisely, the values of `x` and `y` are the same reference (pointer,) and therefore they refer to the same object.

Example (contd.)

```
public class Test
{
  public static void main(String[] args)
  {
    Employee e1 = new Employee(‘‘Adam Smith’’, 80000
    Employee e2 = e1;
    e1.raise_salary(10f);
    System.out.println(e2.salary());
  }
}
```

Example (contd.)



Aliases

- Compare Test with

```
int x1, x2;
x1 = 6;
x2 = x1;
x1 = x1 * 3;
```

- If two variables are aliases, whatever one does to either of them, affects the other, because they refer to the same object.

Shared references

```
public class BankAccount
{
  private float balance;
  public BankAccount(float b) { balance = b; }
  public void deposit(float amount)
  {
    balance = balance + amount;
  }
  public void withdraw(float amount)
  {
    if (balance >= amount)
      balance = balance - amount;
  }
  public float balance() { return balance; }
}
```

Shared references

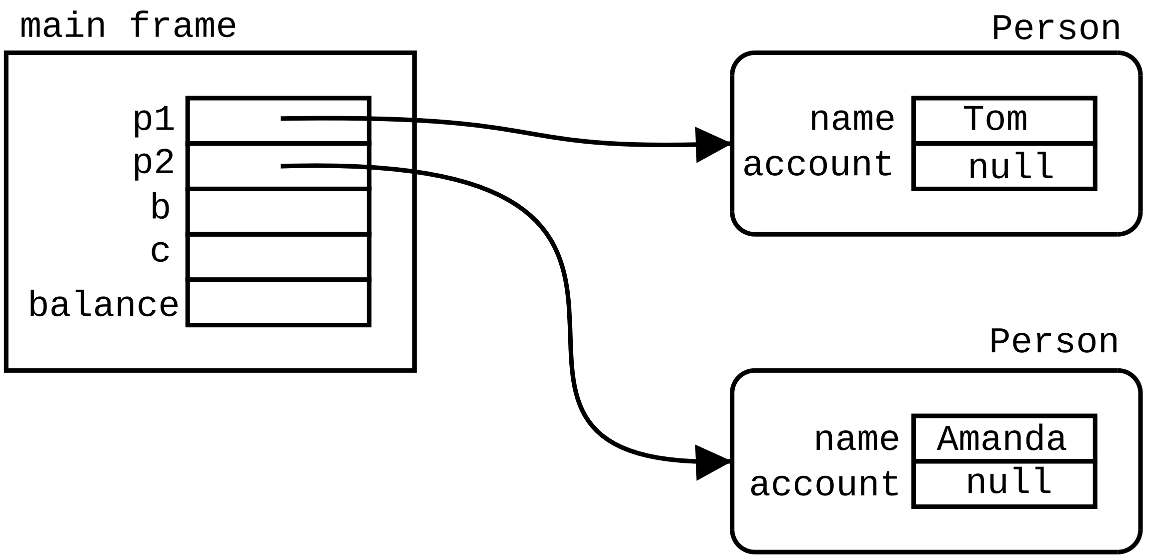
```
public class Person
{
  private String name;
  private BankAccount account;
  public Person(String name) { this.name = name; }
  public void set_account(BankAccount a)
  {
    account = a;
  }
  public String name() { return name; }
  public BankAccount account() { return account; }
}
```

Shared references

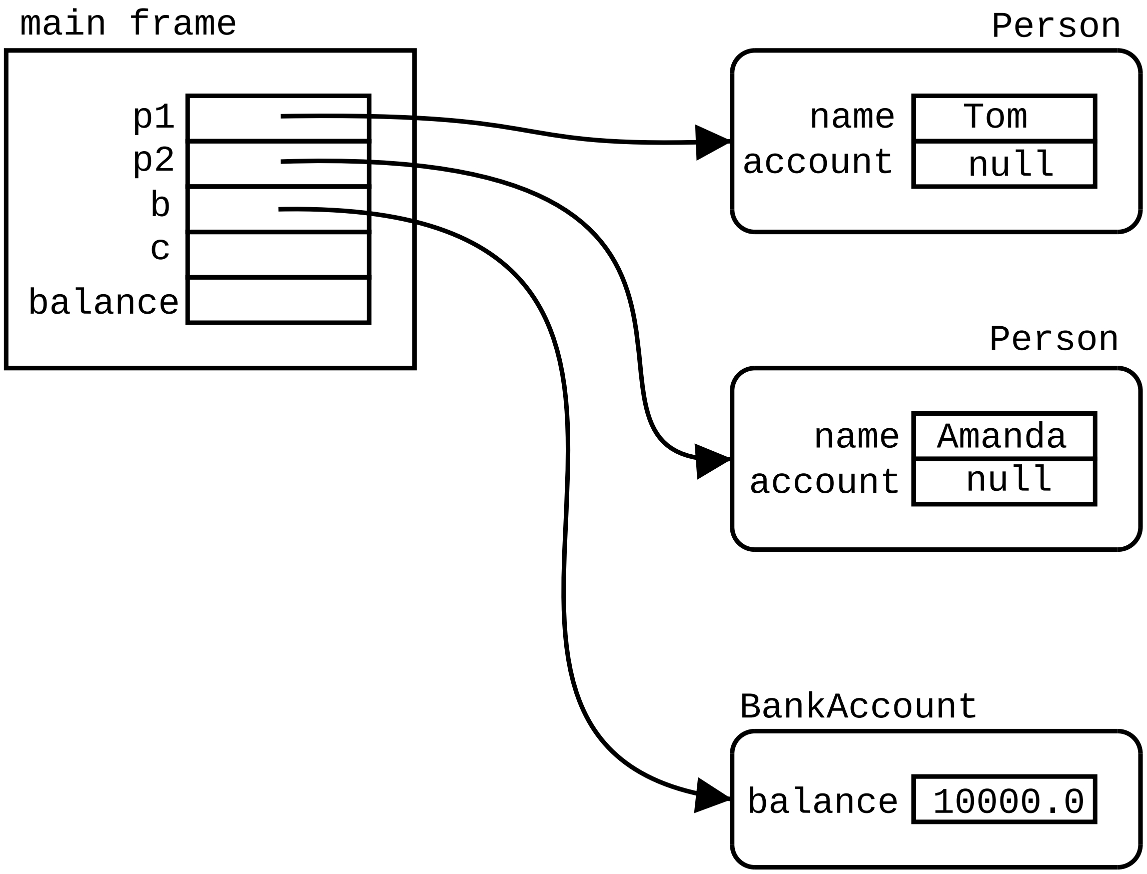
```
public class BankingTest
{
  public static void main(String[] args)
  {
    Person p1 = new Person(“Tom”);
    Person p2 = new Person(“Amanda”);
    BankAccount b = new BankAccount(10000.0f);
    p1.set_account(b);
    p2.set_account(b);

    b.withdraw(500.0f);
    BankAccount c = p2.account();
    float balance = c.balance();
    System.out.println(balance);
  }
}
```

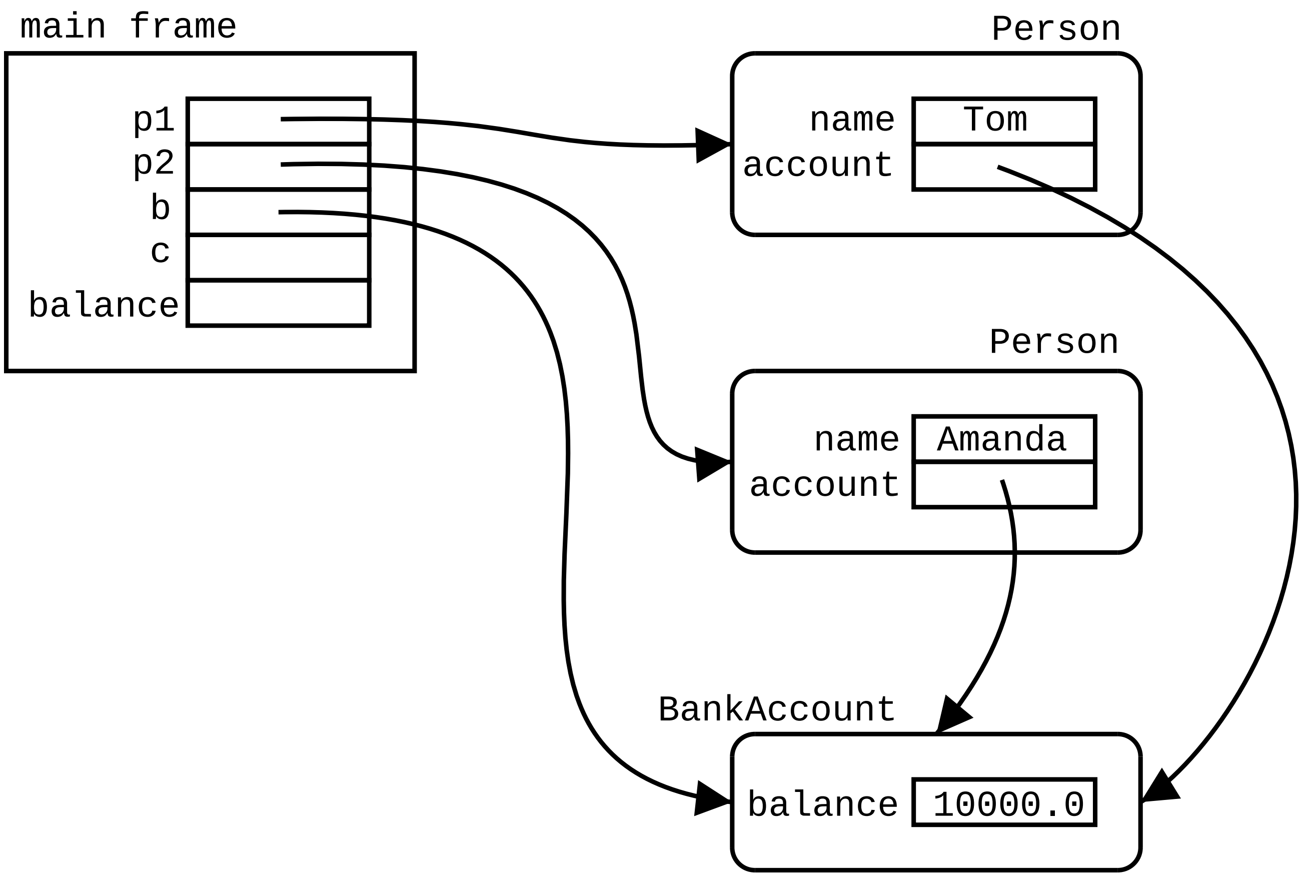
Shared references



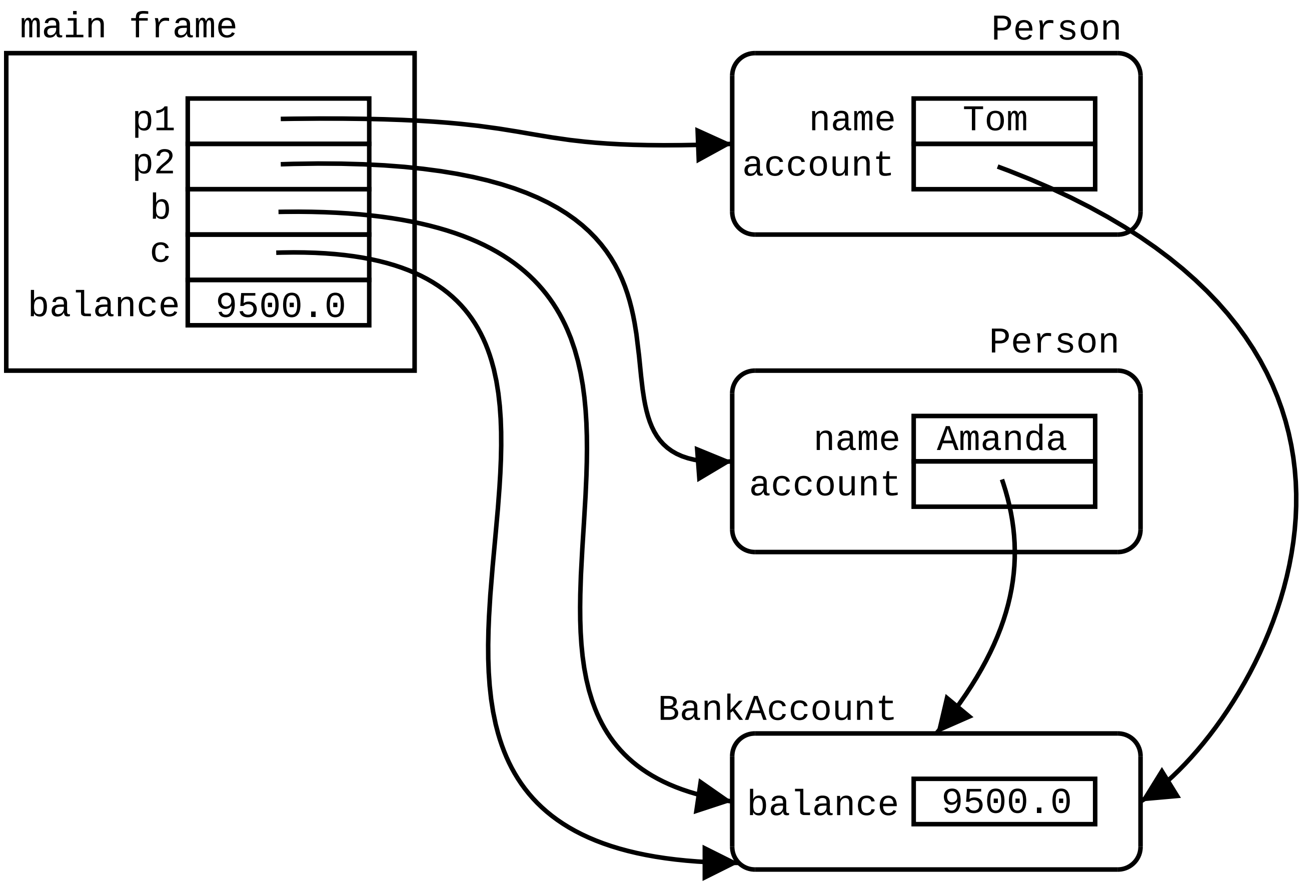
Shared references



Shared references



Shared references



The end