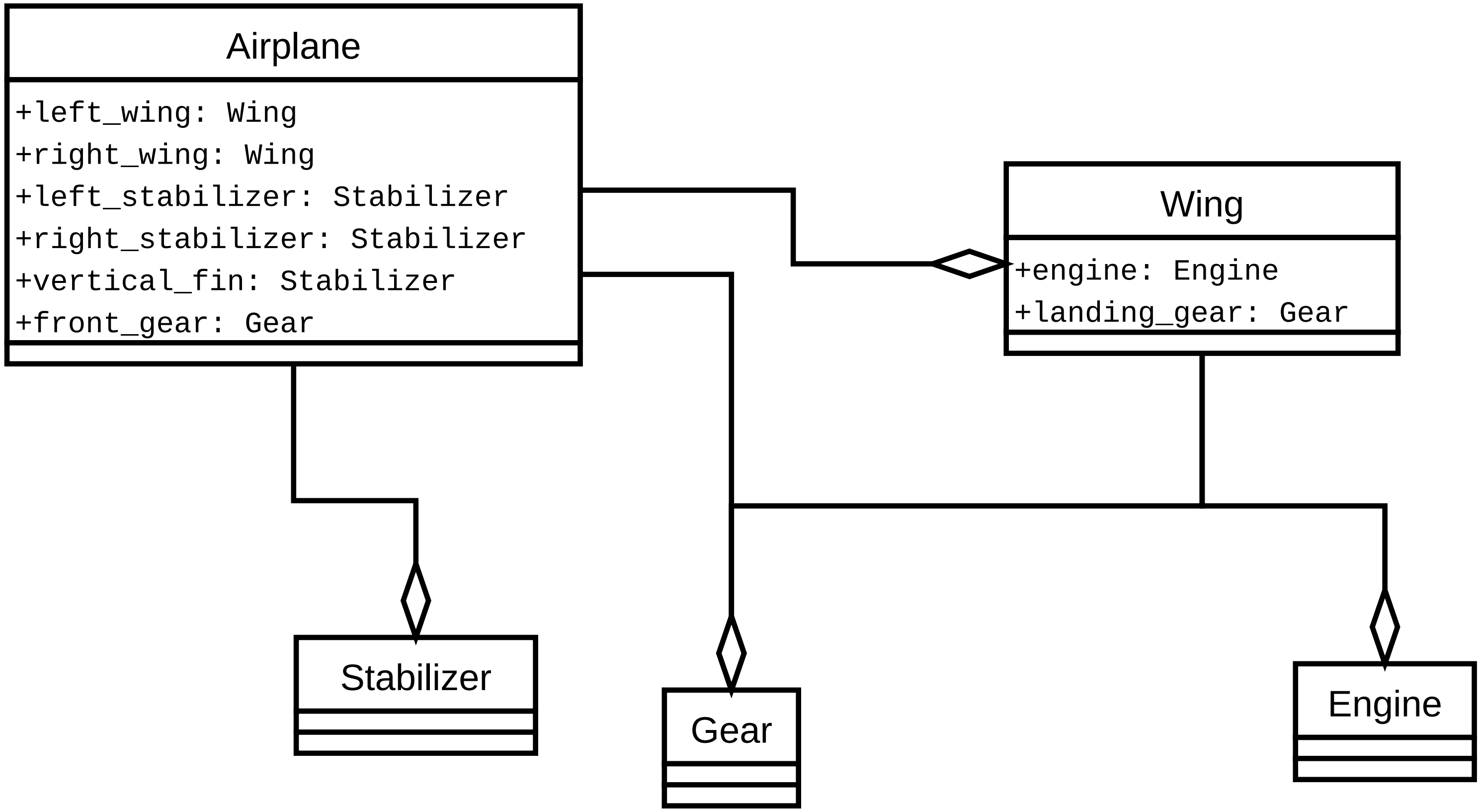
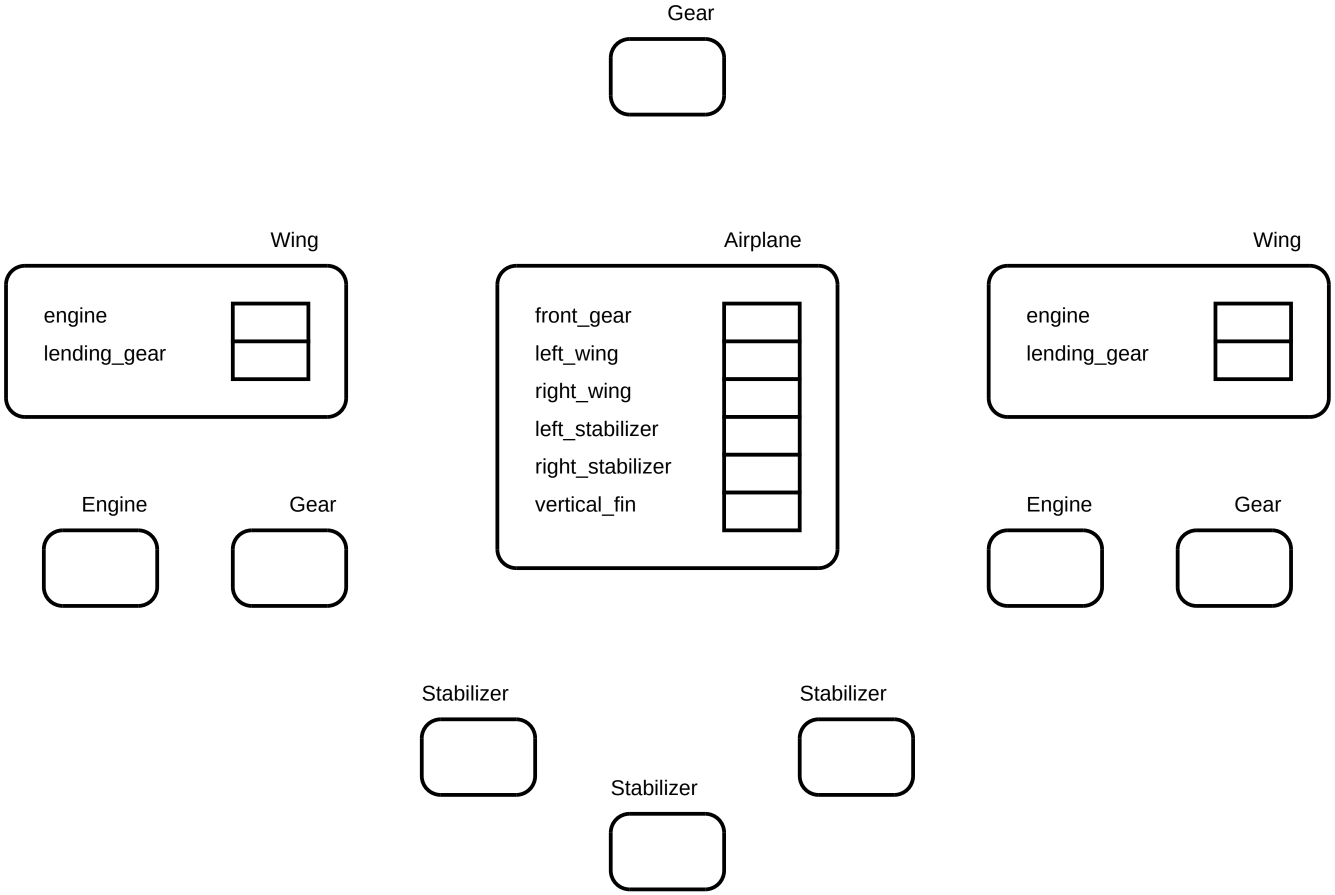
Aggregation



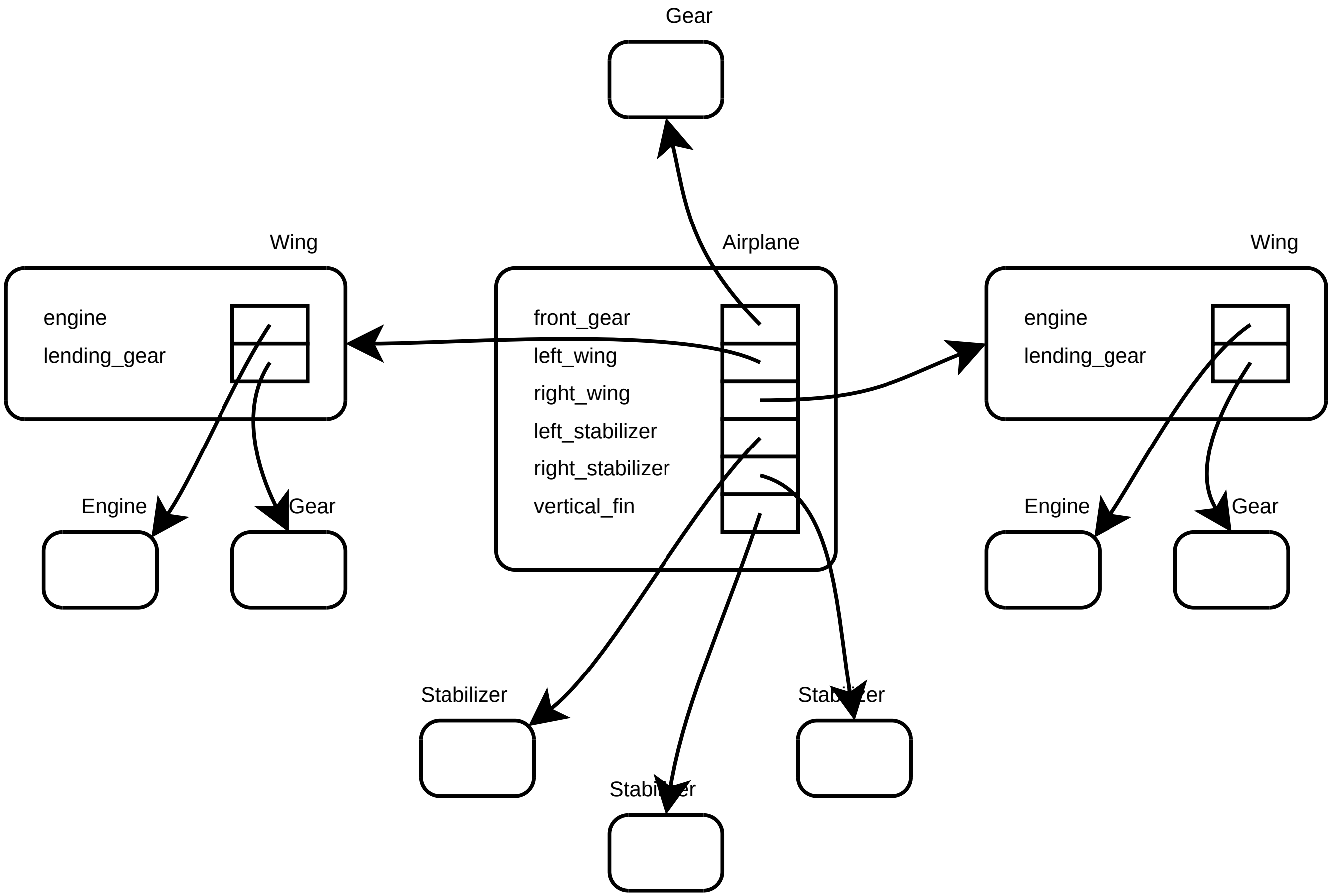
Aggregation

```
public class Airplane
{
    private Wing left_wing, right_wing;
    private Gear front_gear;
    private Stabilizer left_stabilizer;
    private Stabilizer right_stabilizer;
    private Stabilizer vertical_fin;
    //...
}

class Wing
{
    private Engine engine;
    private Gear landing_gear;
    // ...
}

class Engine { ... }
class Gear { ... }
class Stabilizer { ... }
```

Aggregation



Aggregation



Aggregation

```
class Wing
{
    private Engine engine;
    private Gear landing_gear;

    public Wing()
    {
        engine = new Engine();
        landing_gear = new Gear();
    }
}
```

Aggregation

```
public class Airplane
{
    private Wing left_wing, right_wing;
    private Gear front_gear;
    private Stabilizer left_stabilizer;
    private Stabilizer right_stabilizer;
    private Stabilizer vertical_fin;

    public Airplane()
    {
        left_wing = new Wing();
        right_wing = new Wing();
        front_gear = new Gear();
        left_stabilizer = new Stabilizer();
        right_stabilizer = new Stabilizer();
        vertical_fin = new Stabilizer();
    }
}
```

Aggregation

```
public class AirplaneSimulator
{
    public static void main(String[] args)
    {
        Airplane plane1, plane2;
        plane1 = new Airplane();
        plane2 = new Airplane();
    }
}
```

Aggregation

```
class Engine
{
    public void start()
    {
        //...
    }
}
```

Aggregation

```
class Wing
{
    private Engine engine;
    private Gear landing_gear;

    public Wing()
    {
        engine = new Engine();
        landing_gear = new Gear();
    }

    public void startEngine()
    {
        engine.start();
    }
}
```

Aggregation

```
public class Airplane
{
    private Wing left_wing, right_wing;
    private Gear front_gear;
    private Stabilizer left_stabilizer;
    private Stabilizer right_stabilizer;
    private Stabilizer vertical_fin;

    public Airplane()
    {
        // ...
    }
    public void start()
    {
        left_wing.startEngine();
        right_wing.startEngine();
    }
}
```

Aggregation

```
public class AirplaneSimulator
{
    public static void main(String[] args)
    {
        Airplane plane1, plane2;
        plane1 = new Airplane();
        plane2 = new Airplane();
        plane1.start();
        plane2.start();
    }
}
```

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.

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.

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; }
}
```

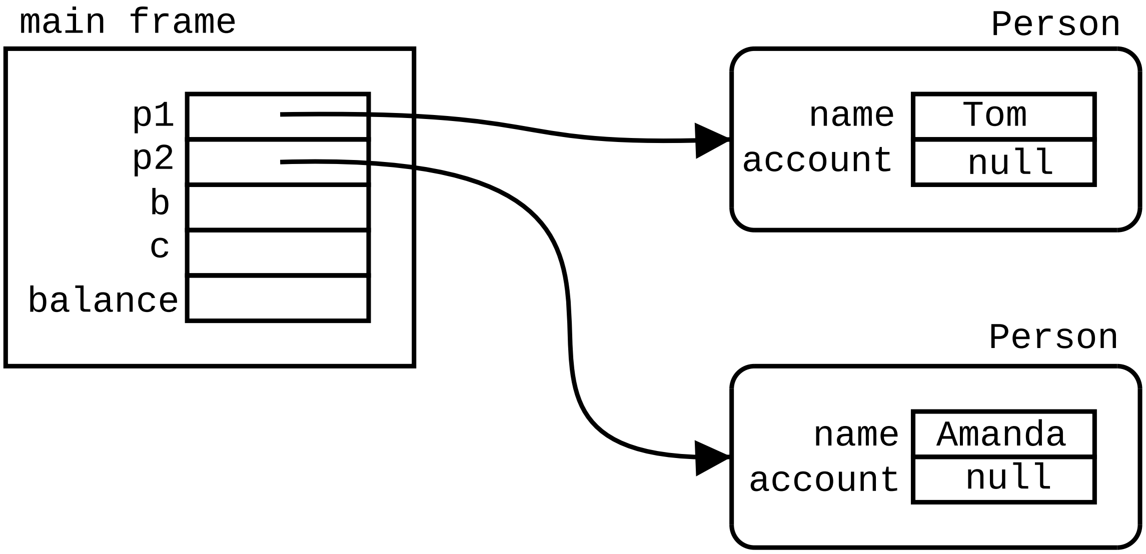
Shared references

```
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; }
}
```

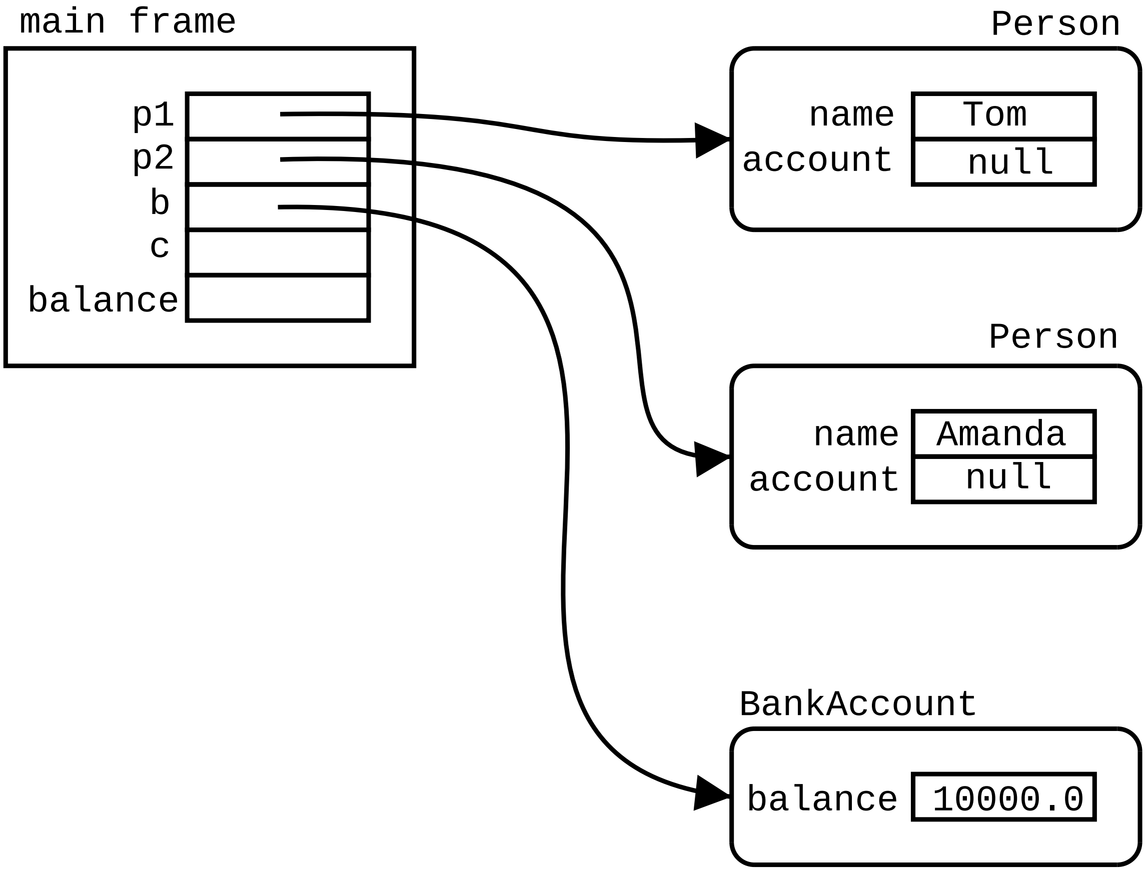
Shared references

```
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);
  }
}
```

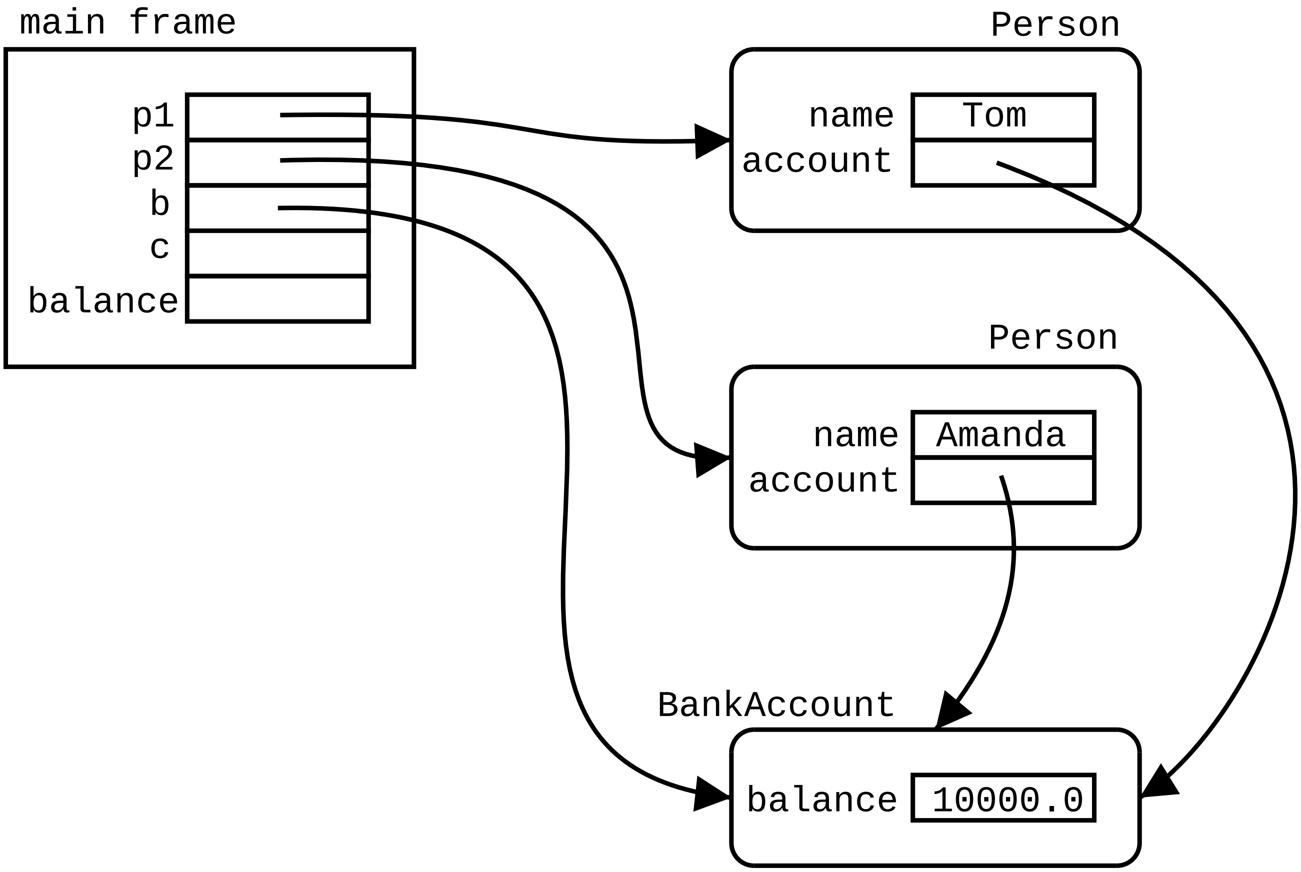
Shared references



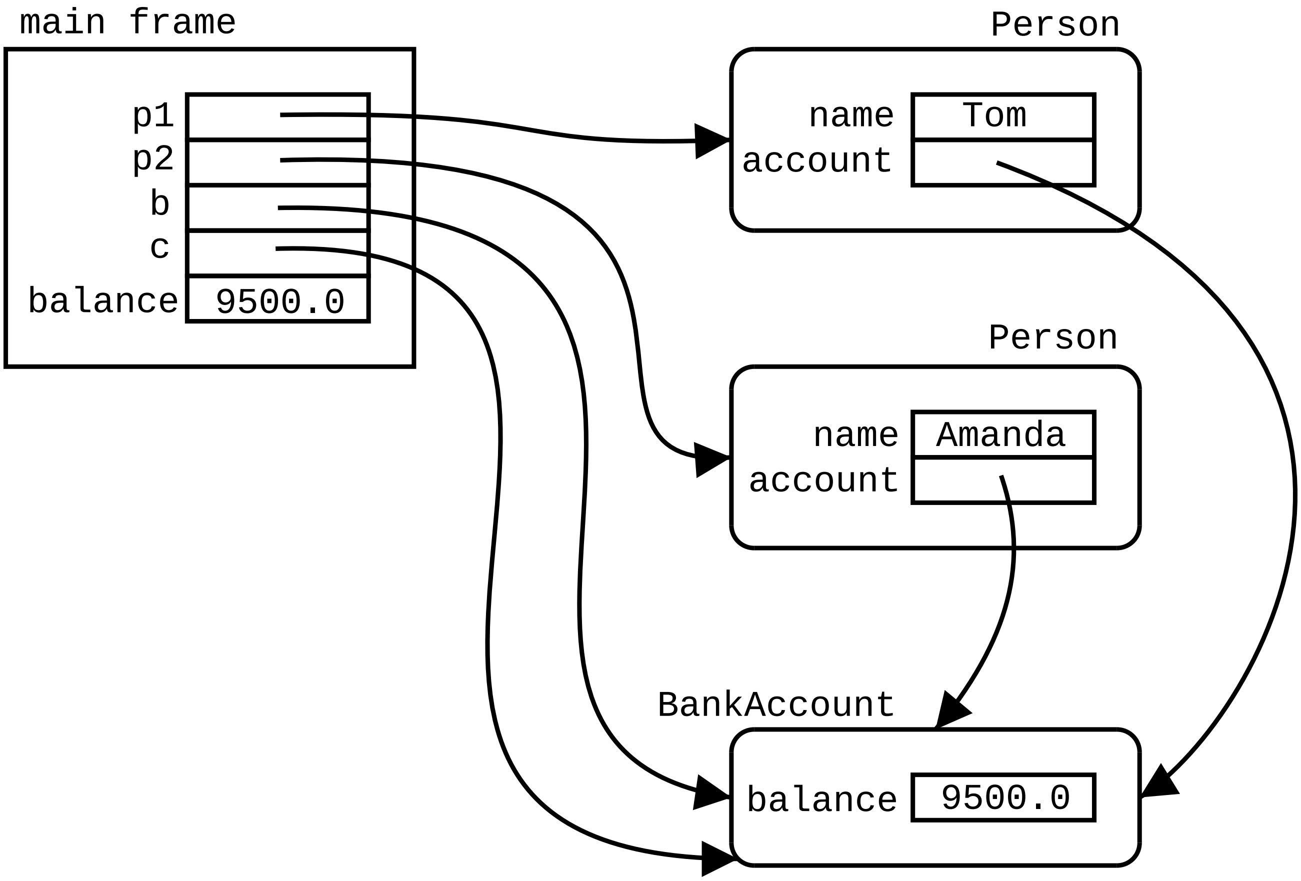
Shared references



Shared references



Shared references



Shared references

```
class A { int n; }

class B { A x; }

class C { A y; }

class D {
  void p()
  {
    B b = new B();
    b.x = new A();
    C c = new C();
    c.y = b.x;
    c.y.n = 3;
    b.x.n = 5;
  }
}
```

Shared references

```
class A { int n; }

class B { A x; }

class C {
  A y;
  C(A z) { y = z; }
}

class D {
  void p()
  {
    B b = new B();
    b.x = new A();
    C c = new C(b.x);
    c.y.n = 3;
    b.x.n = 5;
  }
}
```

Pointer equality

- Pointer equality also called “physical” equality is equality (sameness) of references.

- The == operator is used for testing for pointer equality.

- Pointer equality is used to test for sameness of objects:

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

- ...then `x == y` is true, but in

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

- ... `x == y` is false, even if the attributes of the objects are the same.

- Pointer equality is an equivalence between objects of the same class only.

Example

```
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);

    BankAccount d = p1.account();
    d.withdraw(500.0f);
    BankAccount c = p2.account();
    if (c == d)
      System.out.println(“It’s a shared account”);
  }
}
```

Being equal to something

- Structural equality: when the aggregates (parts) of two different objects are equal

- Structural equality is only between objects of the same class.

- Two objects are structurally equal if their attributes are equal

- Suppose we have a class

```
class A {
    String x, y;
    A(String x, String y)
    {
        this.x = x;
        this.y = y;
    }
}
```

Being equal to something

- and there is some client with

```
A a1 = new A(“hello”, “bye”);
A a2 = new A(“hello”, “bye”);
A a3 = new A(“bonjour”, “bye”);
```

- then a1 is structurally equal to a2, but a3 is not structurally equal to either a1 or a2.
- If we want to test for structural equality we must explicitely provide the code. This is usually done by writing a method called “equal” or “equals”:

Structural equality

```
class A {
    String x, y;
    A(String x, String y)
    {
        this.x = x;
        this.y = y;
    }
    boolean equals(A other)
    {
        return this.x == other.a
            && this.y == other.y;
    }
}
```

Structural equality

```
public class Test
{
  public static void main(String[] args)
  {
    A a1 = new A("hello", "bye");
    A a2 = new A("hello", "bye");
    A a3 = new A("bonjour", "bye");
    if (a1.equals(a2))
      System.out.println(“a1 is equal to a2”);
    if (a2.equals(a3))
      System.out.println(“a2 is equal to a3”);
    if (a1 == a2)
      System.out.println(“a1 is the same as s2”);
  }
}
```

Structural equality vs pointer equality

- Note that
  - If two objects are the same (equal by pointer equality) then they are (structurally) equal, ...
    This is, x == y implies that x`.equals(`y`)` must evaluate to true.
  - ...but if two objects are structurally equal, they may not be physically the same.
    This is, it may be the case that x`.equals(`y`)` evaluates to true, but x == y may be false.

Example

```
public class BankAccount {
  private float balance;
  // ... same as before
  public boolean equals(BankAccount other_account)
  {
    return this.balance == other_account.balance;
  }
}
```

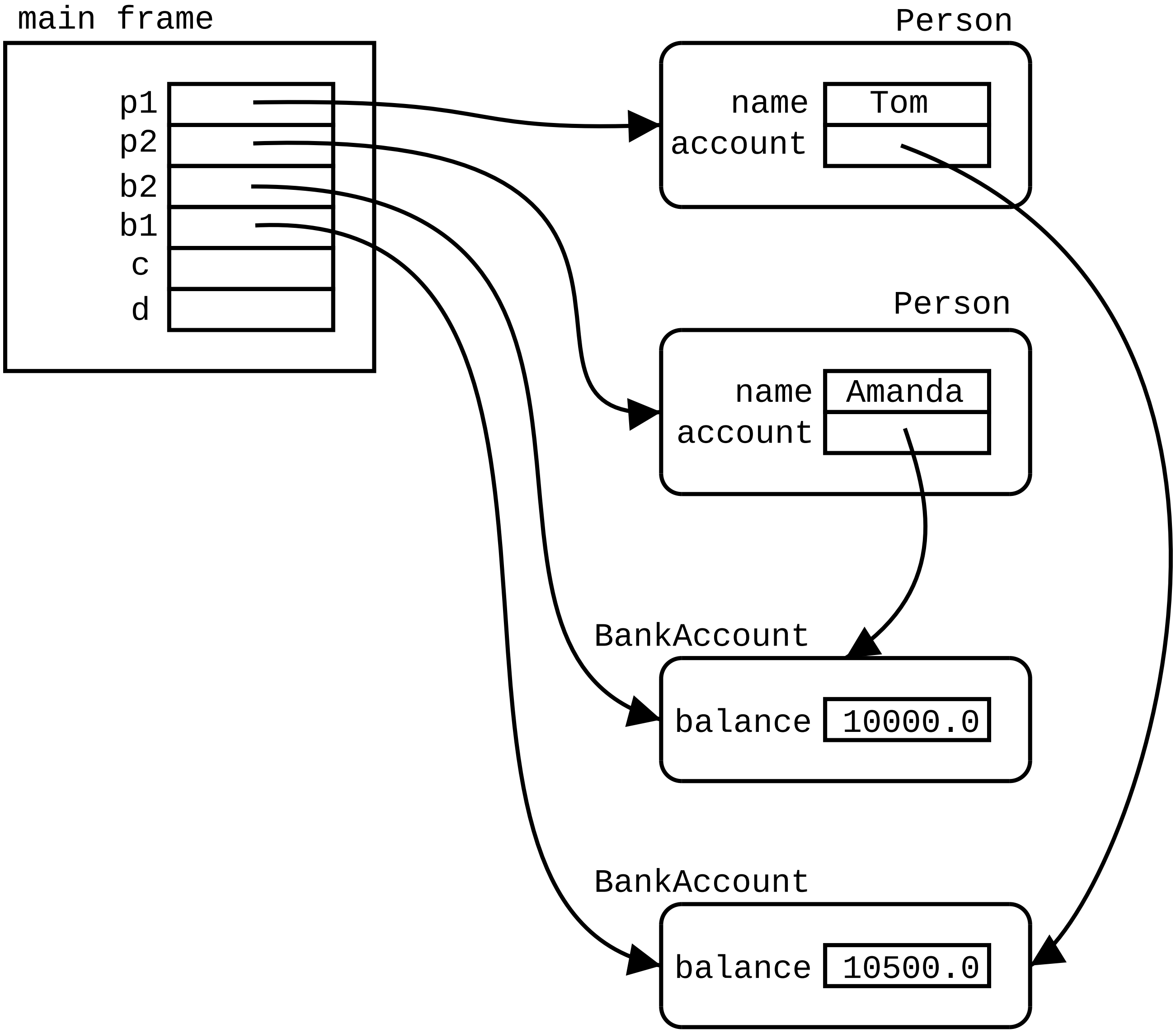
Example

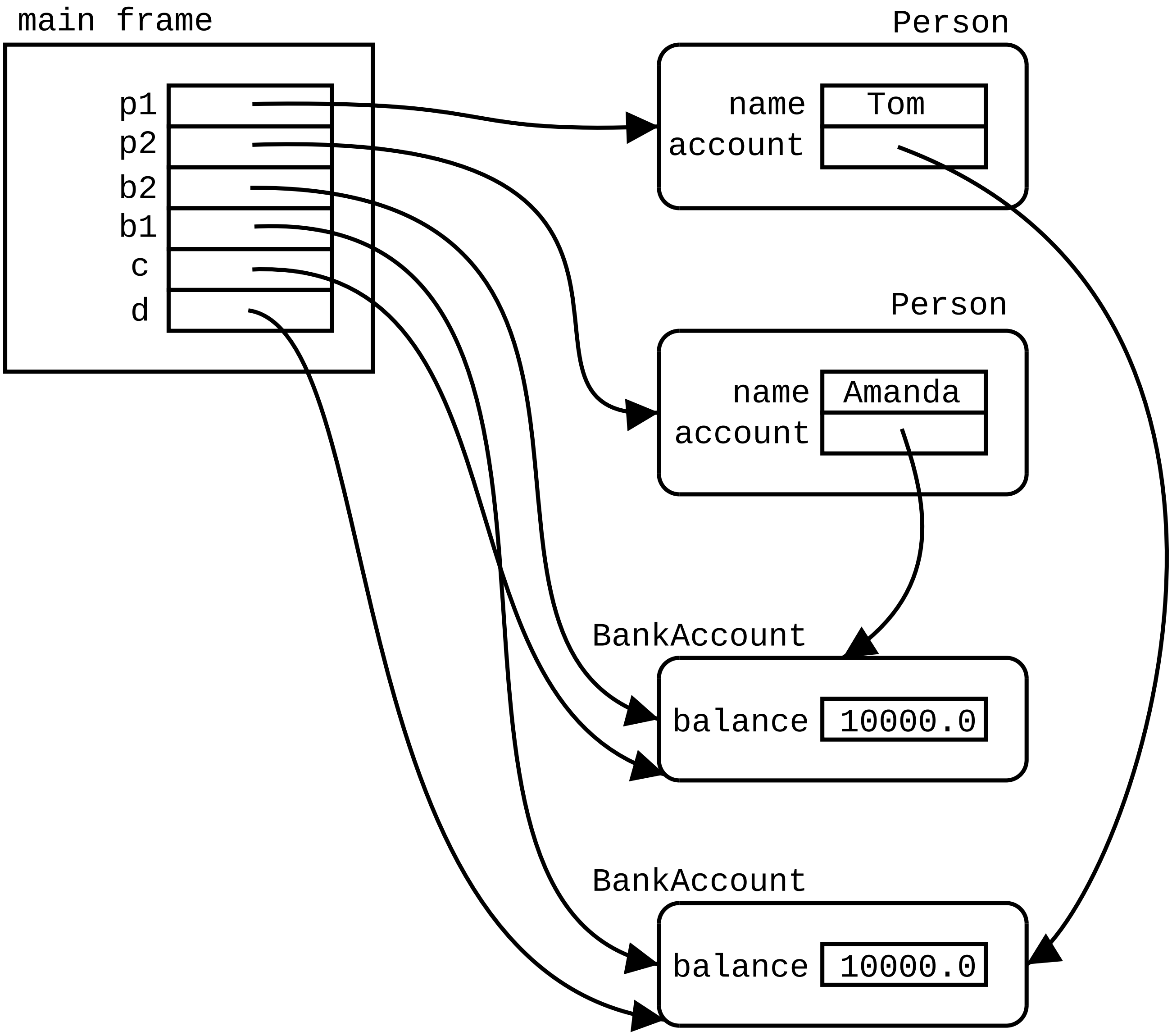
```
public class BankAccount {
  private float balance;
  // ... same as before
  public boolean equals(BankAccount other_account)
  {
    return this.balance == other_account.balance;
  }
}
```

Example

```
public class BankingTest
{
  public static void main(String[] args)
  {
    Person p1 = new Person(“Tom”);
    Person p2 = new Person(“Amanda”);
    BankAccount b1 = new BankAccount(10500.0f);
    BankAccount b2 = new BankAccount(10000.0f);
    p1.set_account(b1);
    p2.set_account(b2);
    BankAccount d = p1.account();
    d.withdraw(500.0f);
    BankAccount c = p2.account();
    if (c.equals(d))
      System.out.println(“They are equal accounts”
  }
}
```

Example


main frame
p1
p2
b2
b1
c
d
Person
name
Tom
account
Person
name
Amanda
account
BankAccount
balance
10000.0
BankAccount
balance
10500.0


main frame
p1
p2
b2
b1
c
d
Person
name
Tom
account
Person
name
Amanda
account
BankAccount
balance
10000.0
BankAccount
balance
10000.0

The end