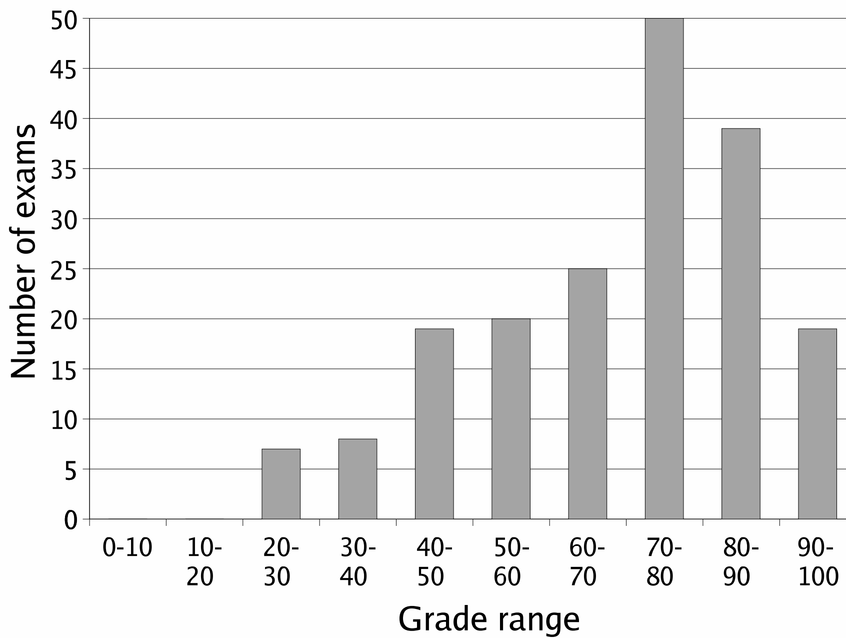
Midterm

Section	Average
1	68.66
2	64.56
3	73.35
Total	64.73

- Median: 71



Defining characteristics of OOP

- A programming language is object-oriented if it supports:
  - Class definitions and class instantiation
  - Message-passing
  - Aggregation
  - Encapsulation
  - Polymorphism
  - Inheritance

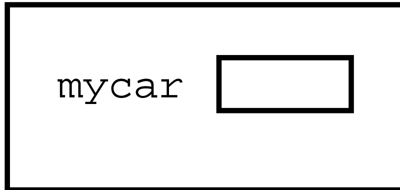
Aggregation: the "has-a" relationship

- Silogisms:
  - **If** every city *has a* mayor, **and** Edinburgh *is a* city, **then** Edinburgh *has a* mayor.
  - **If** every car *has an* engine, **and** this *is a* car, **then** this *has an* engine.
  - **If** every A *has a* B, **and** x *is an* A, **then** x *has a* B.
- In OOP:
  - **If** every object of type A *has an* attribute of type B **and** x *is an* A object **then** x *has an* attribute of type B.
  - **If** a class A *has an* attribute of class B, **and** x *is an* instance of A, **then** x *has an* attribute of class B.

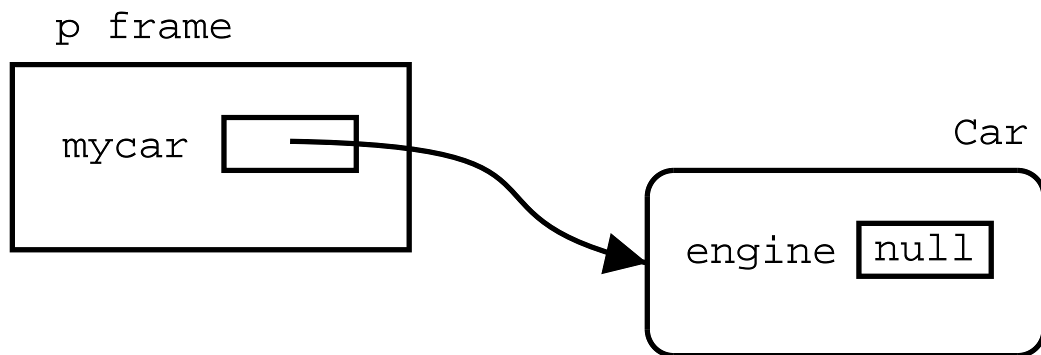
Objects and Aggregation

```
public class Engine {
    // ...
}
public class Car {
    Engine engine;
    // ...
}
public class Something {
    void p()
    {
        Car mycar = new Car();
        mycar.engine = new Engine();
    }
}
```

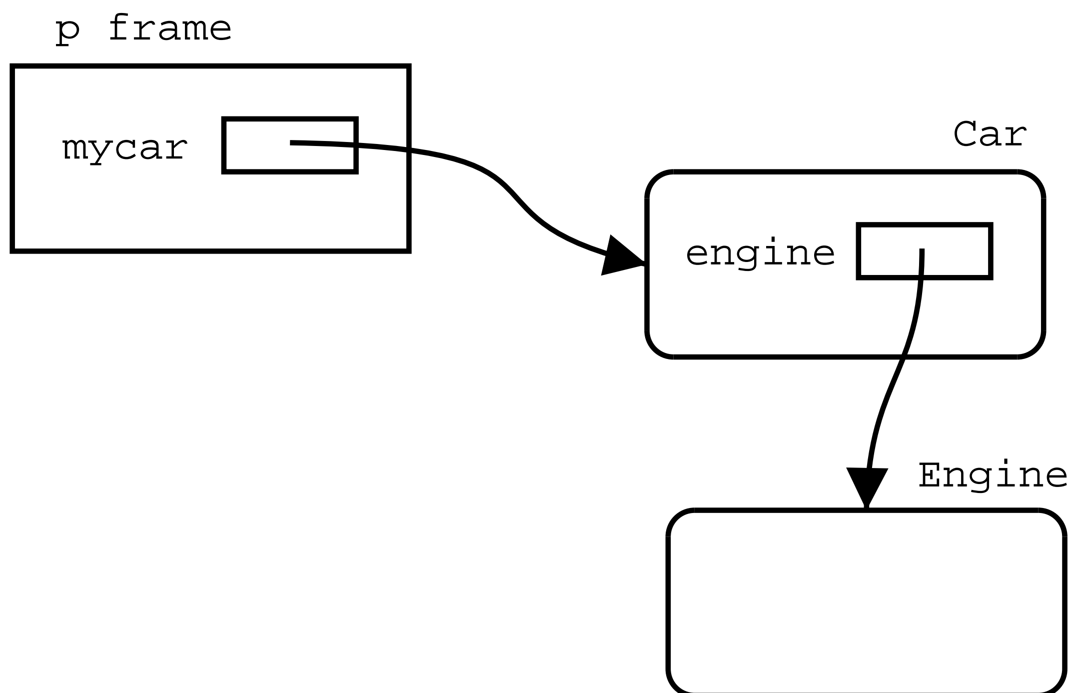
Objects and Aggregation



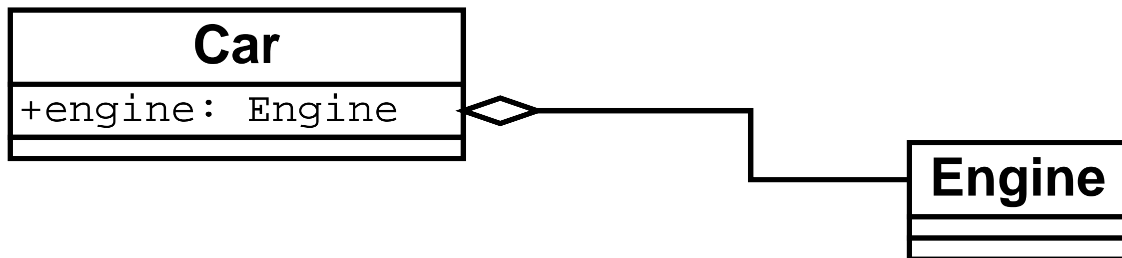
Objects and Aggregation



Objects and Aggregation

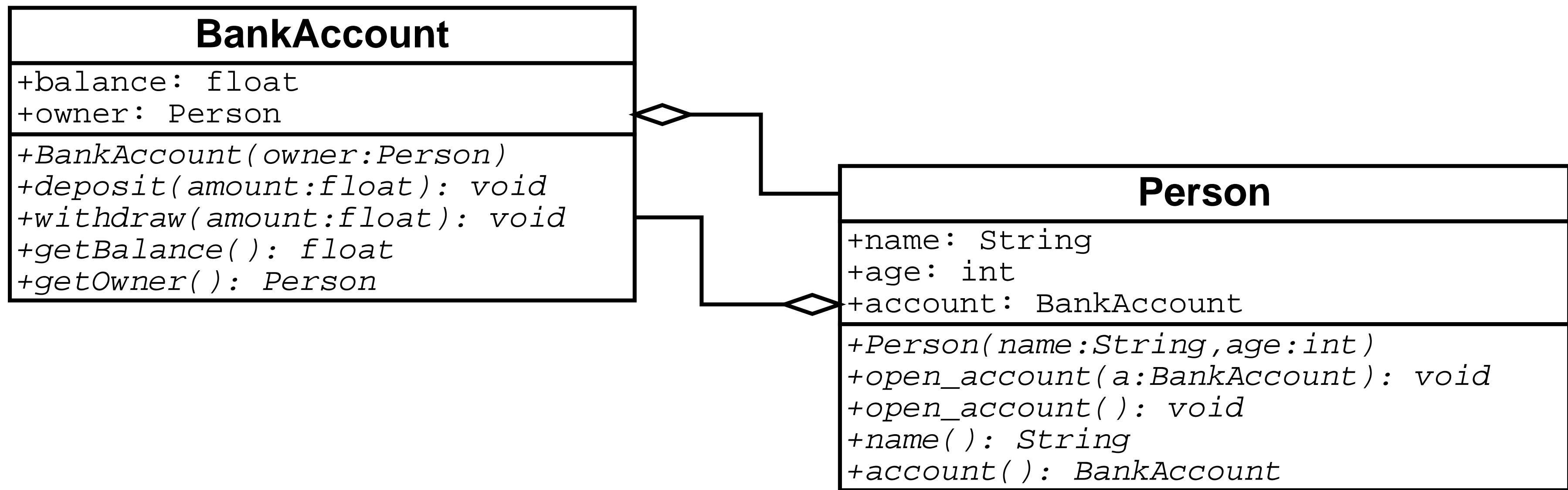


Objects and Aggregation



Mutual references

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



McGill

Mutual reference

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

    public BankAccount(Person owner)
    {
        this.owner = owner;
        balance = 0.0;
    }
    // Etc...
}
```

Mutual reference

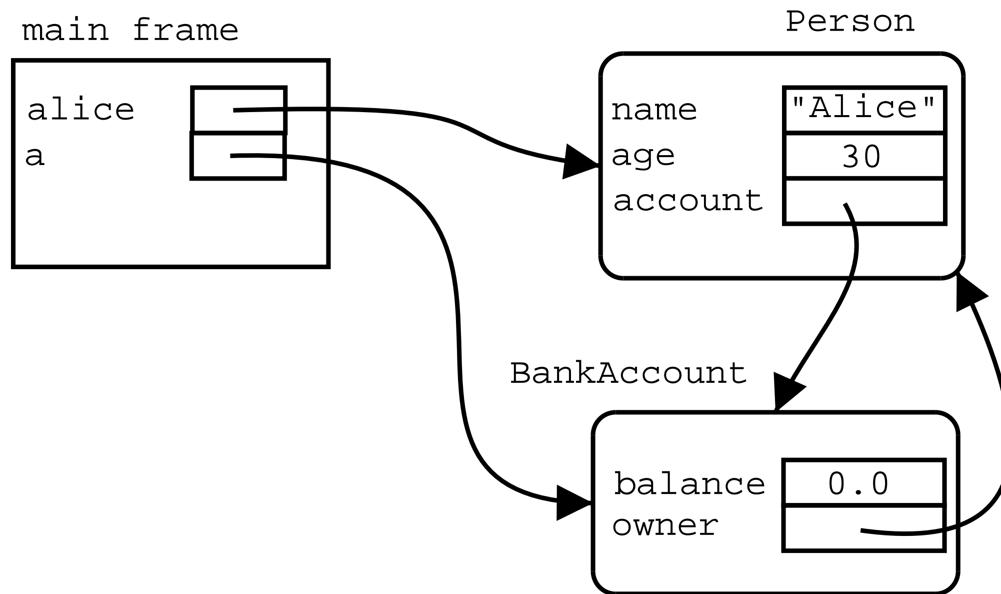
```
public class Person
{
    private String name;
    private int age;
    private BankAccount account;

    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);
    }
}
```

Mutual reference

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

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);
    }
}
```

Aliases

- 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.
- 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.

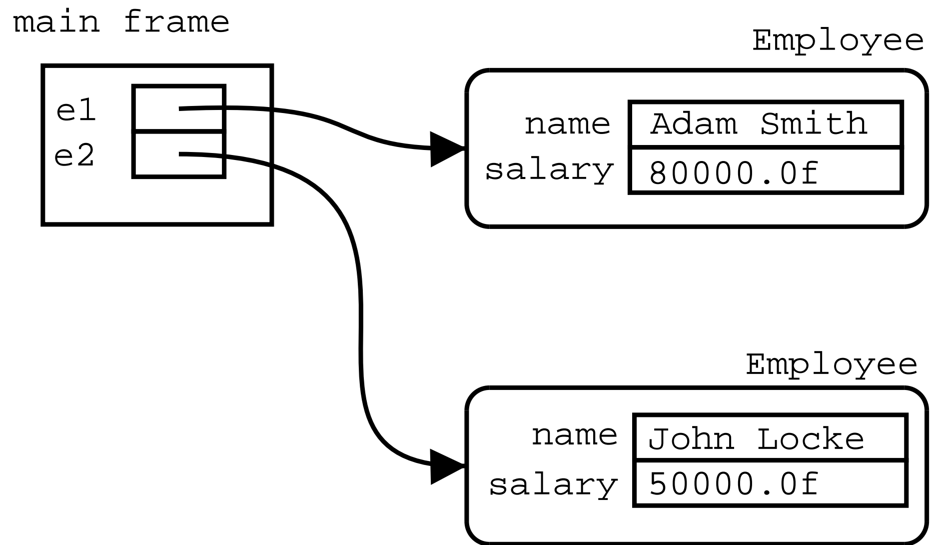
Example:

```
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);
    }
}
```

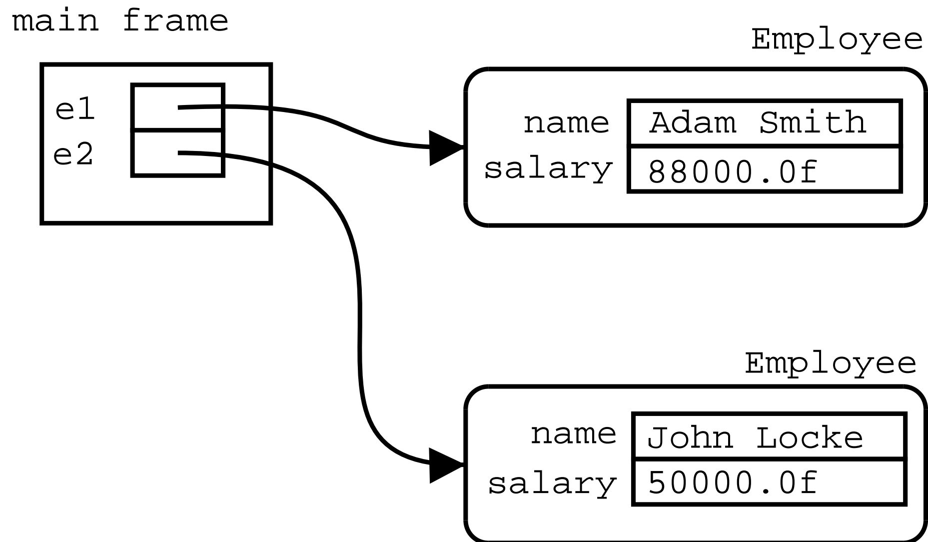
Example (contd.)

```
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.)



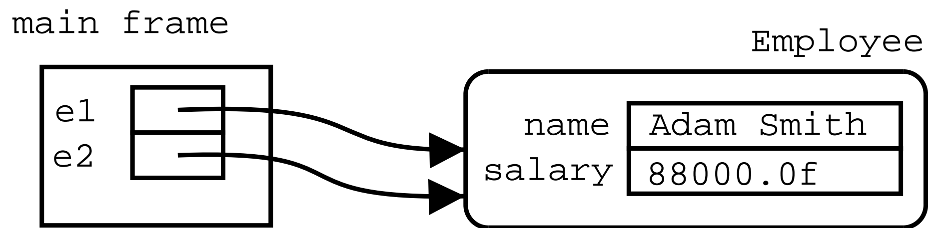
Example (contd.)



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

- Representing:
  - Shared resources
  - Shared information
  - Shared parts
- Example: shared bank account
- Done by creating aliases in different objects

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

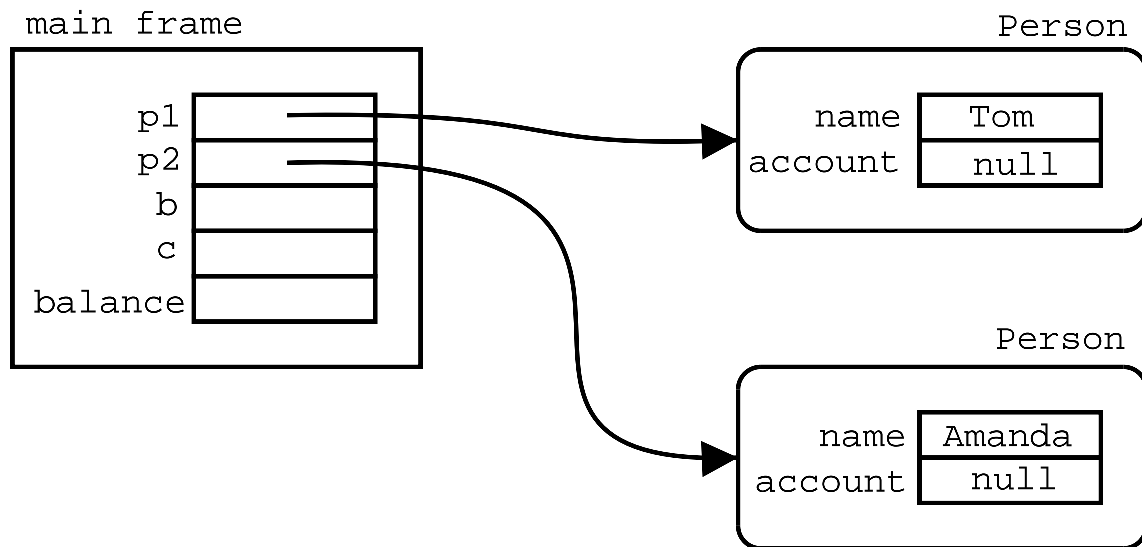
```
public class Person
{
  private String name;
  private BankAccount account;
  public Person(String name) { this.name = name; }
  public void open_account(BankAccount a)
  {
    account = a;
  }
  public String name() { return name; }
  public BankAccount account() { return account; }
}
```

Shared references

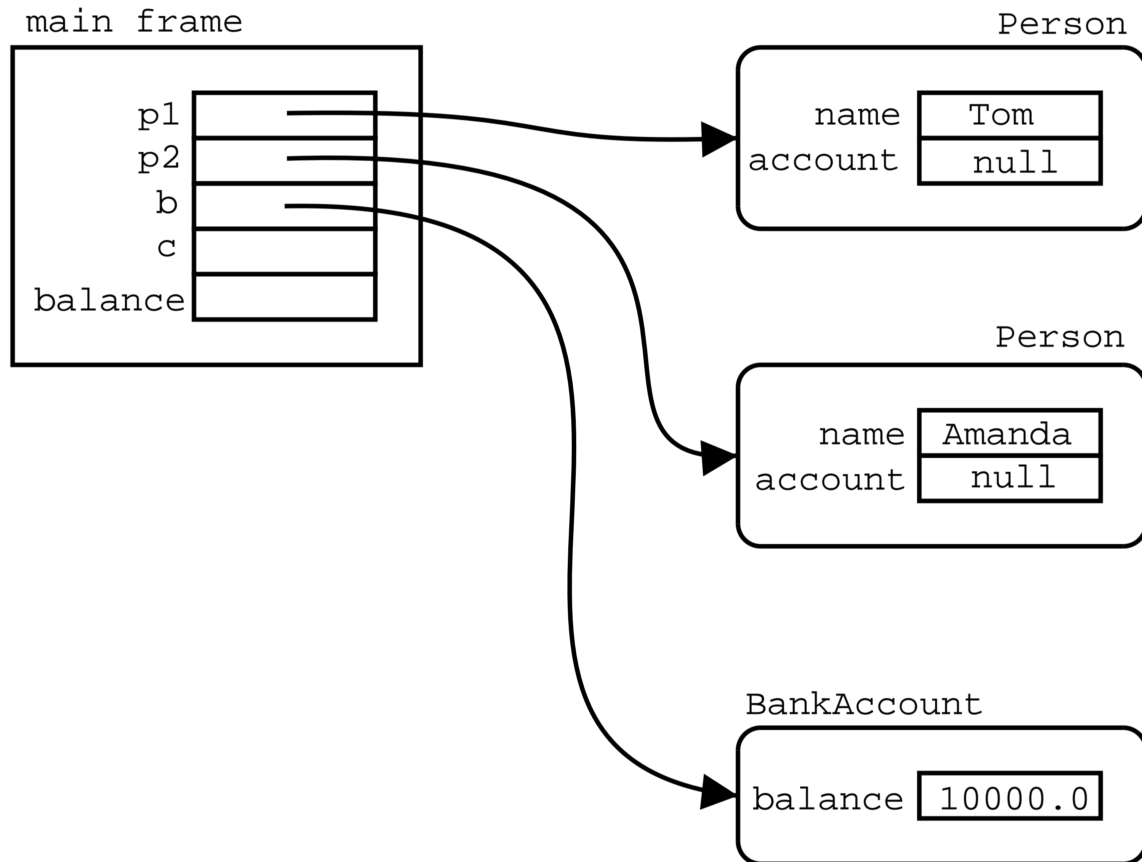
```
public class BankingTest
{
  public static void main(String[] args)
  {
    Person p1 = new Person(“Tom”);
    Person p2 = new Person(“Amanda”);
    BankAccount b = new BankAccount(10000.0f);
    p1.set_account(b);
    p2.set_account(b);

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

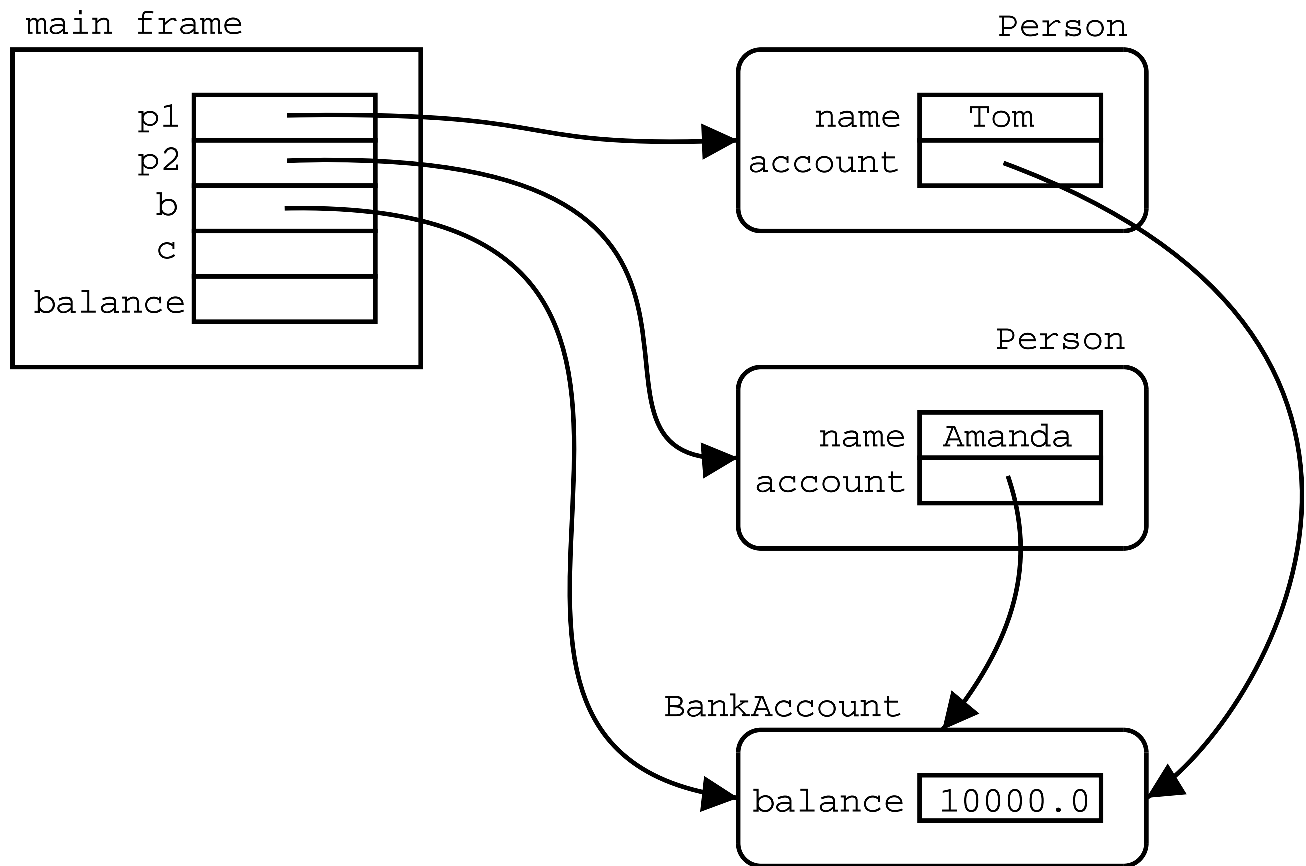
Shared references



Shared references



Shared references



Shared references vs static variables

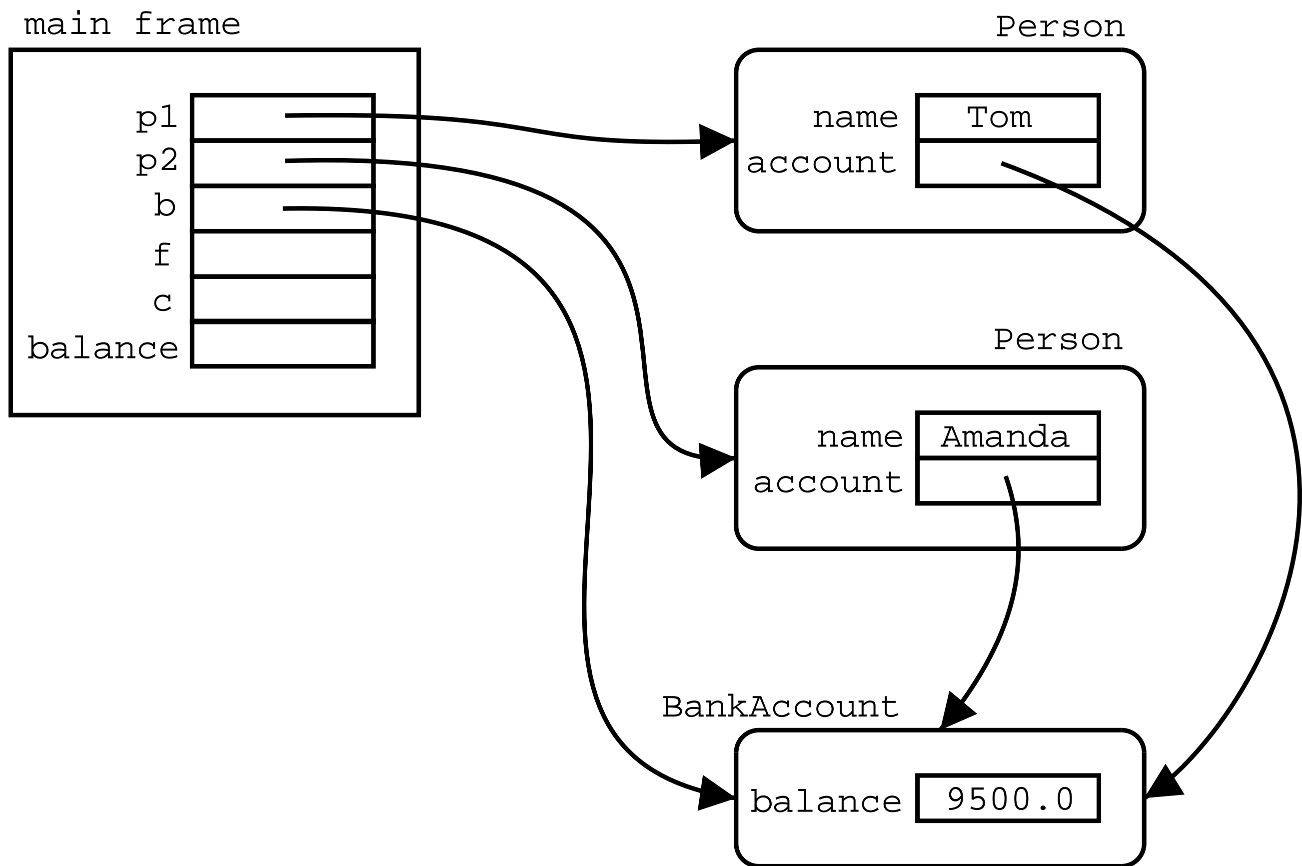
Shared references vs static variables

- In the BankingTest example b is shared between p1 and p2 only, not between all Person objects
- Static variables are like aliases, but they force all objects of the class to share the static reference, while non-static shared references are shared between specific objects.
- Furthermore, if a variable is declares as static the object it refers to is always shared between all objects in the class, while a non-static shared reference might become "unshared".

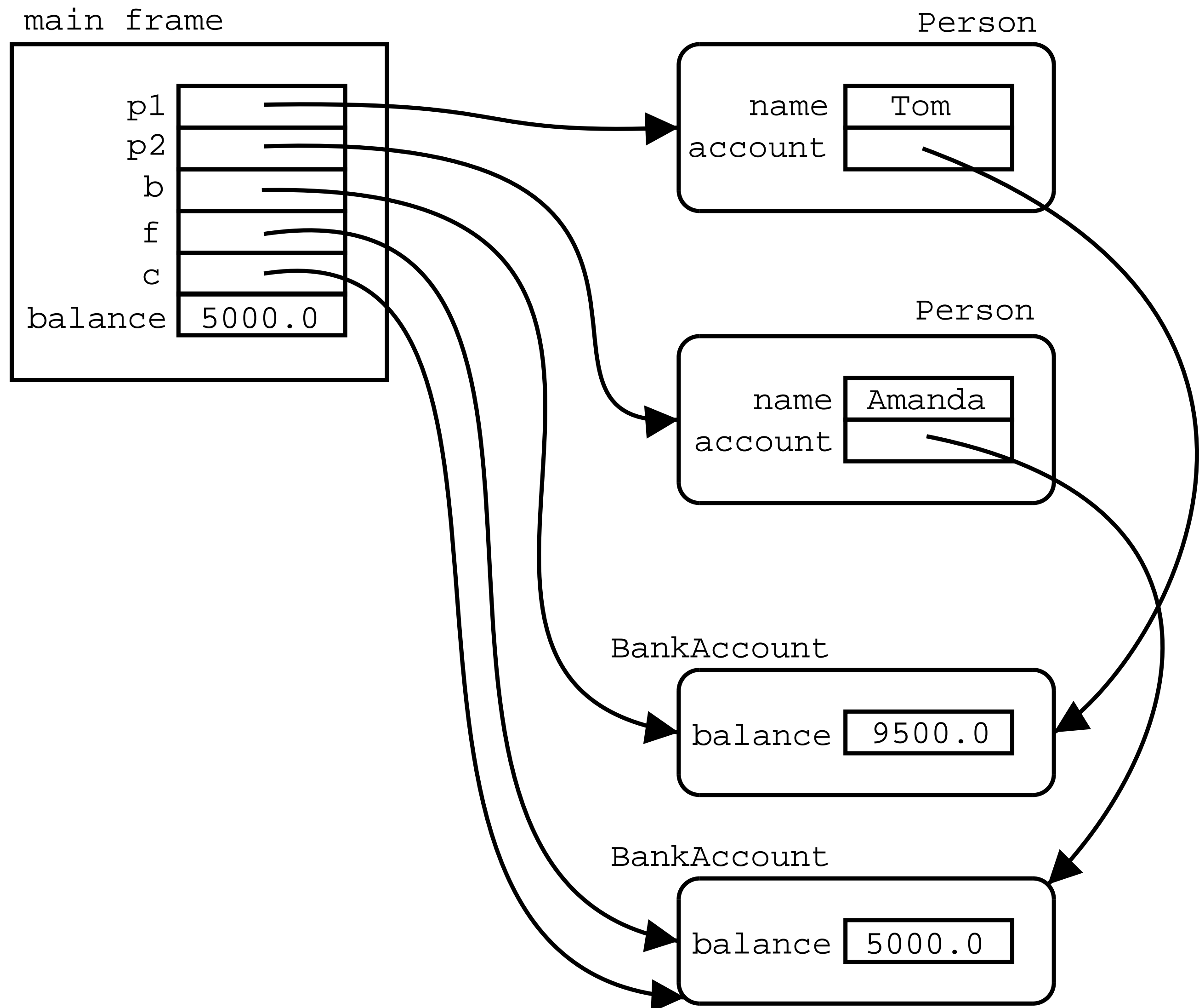
Shared references vs static variables

```
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.open_account(b);
    p2.open_account(b);
    b.withdraw(500.0f);
    BankAccount f = new BankAccount(5000.0f);
    p2.open_account(f);
    BankAccount c = p2.account();
    float balance = c.balance();
    System.out.println(balance);
  }
}
```

Shared references vs static variables



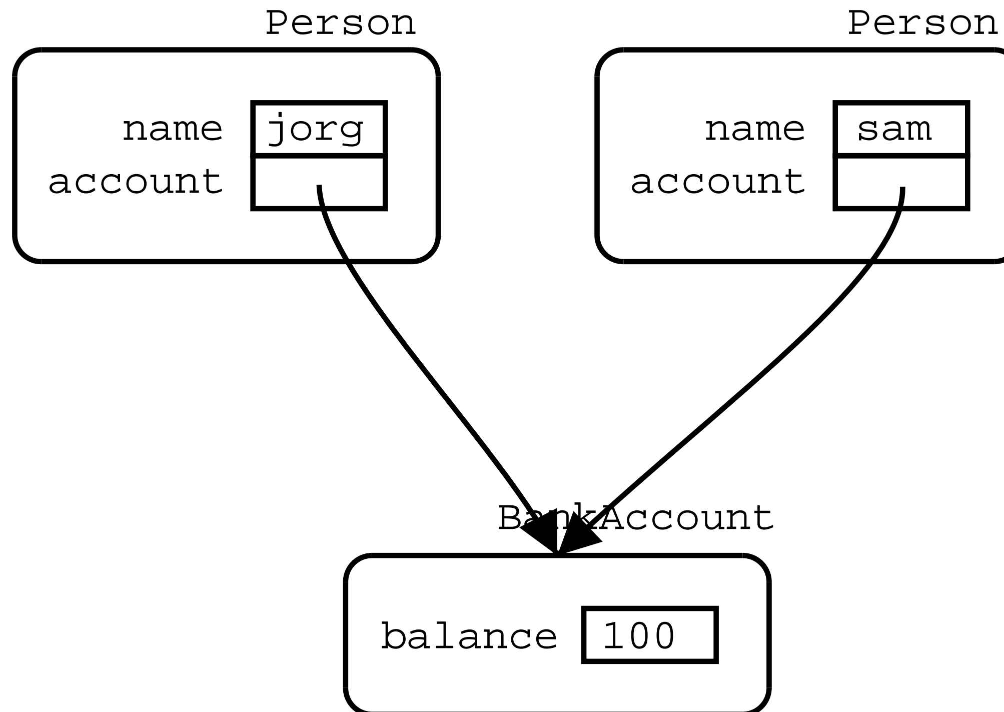
Shared references vs static variables



Equality between objects

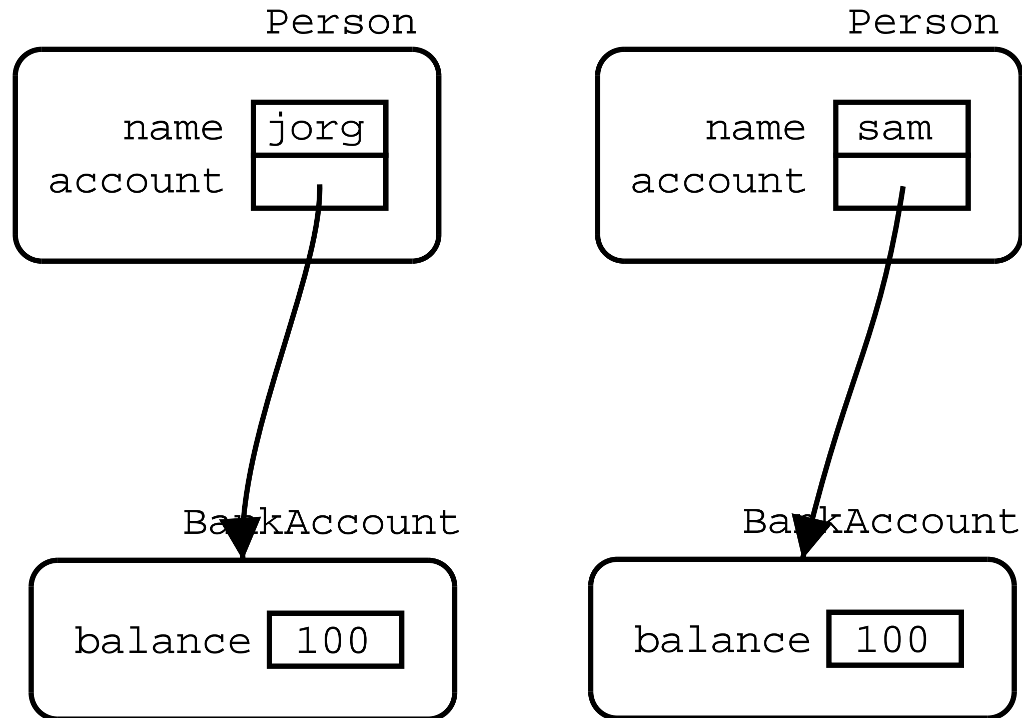
- To check whether two primitive values are equal we use ==
- How do we know if two objects are equal?
- What does it mean to say that two objects are equal?
- *Equality is not the same as "sameness".*
- Possible questions related to equality:
  - Given two people, do they share a bank account?
  - Given two people, do they have the same amount of money in their bank accounts?

Equality between objects



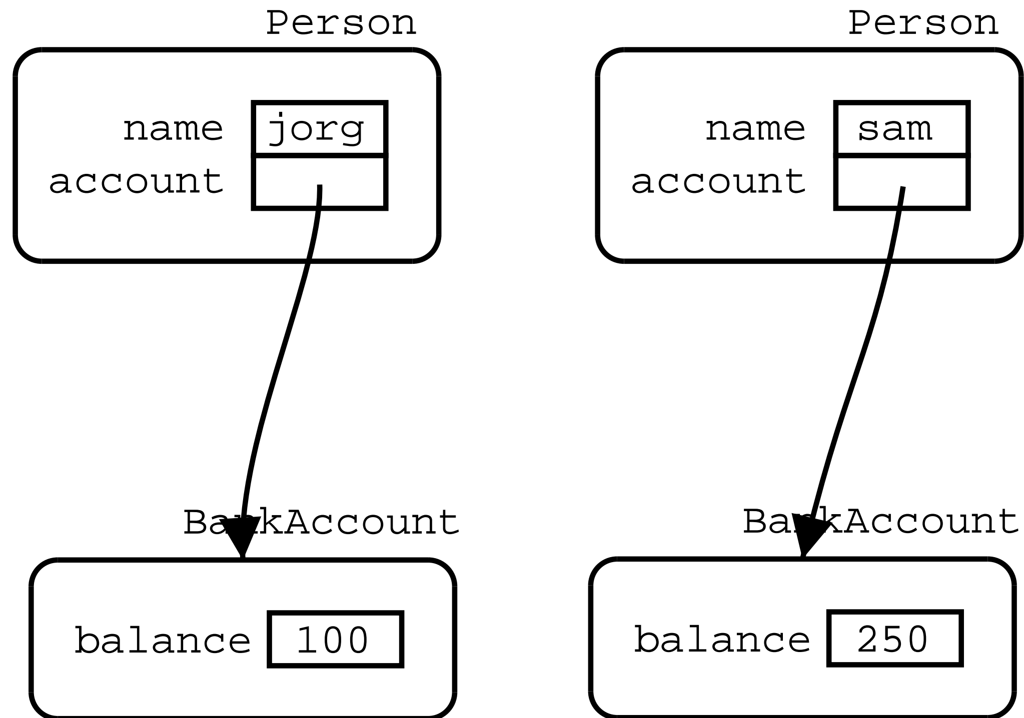
Jorg's account and Sam's account are *pointer equal*, i.e. they are the same.

Equality between objects



Jorg's account and Sam's account are *structurally equal*, i.e. the values of the attributes are the equal.

Equality between objects



Jorg's account and Sam's account are *different*, i.e. the object is not shared and values of the attributes are the different.

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.

McGill

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.open_account(b);
    p2.open_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;
    }
}
```

McGill

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.

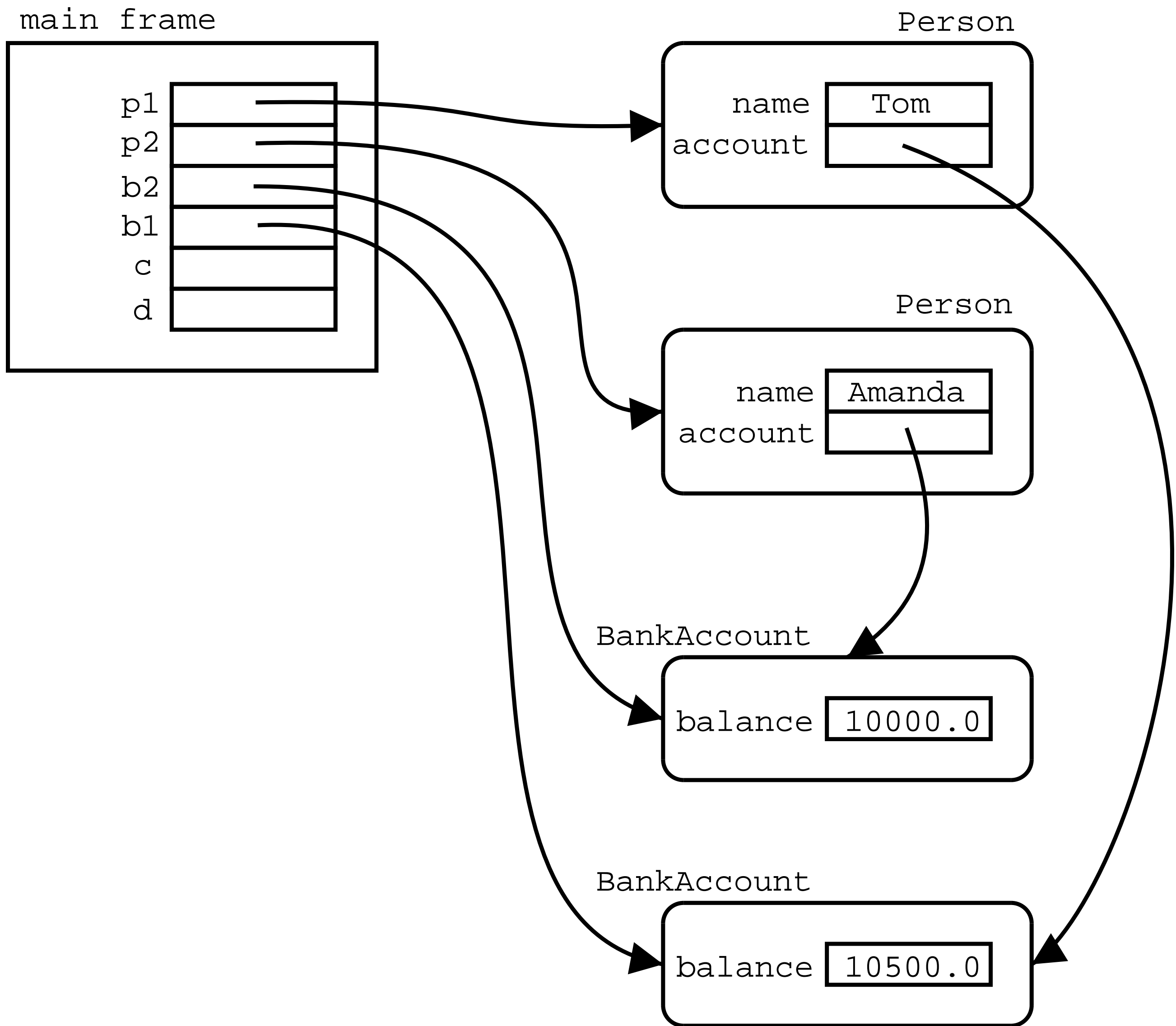
Example

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

Example

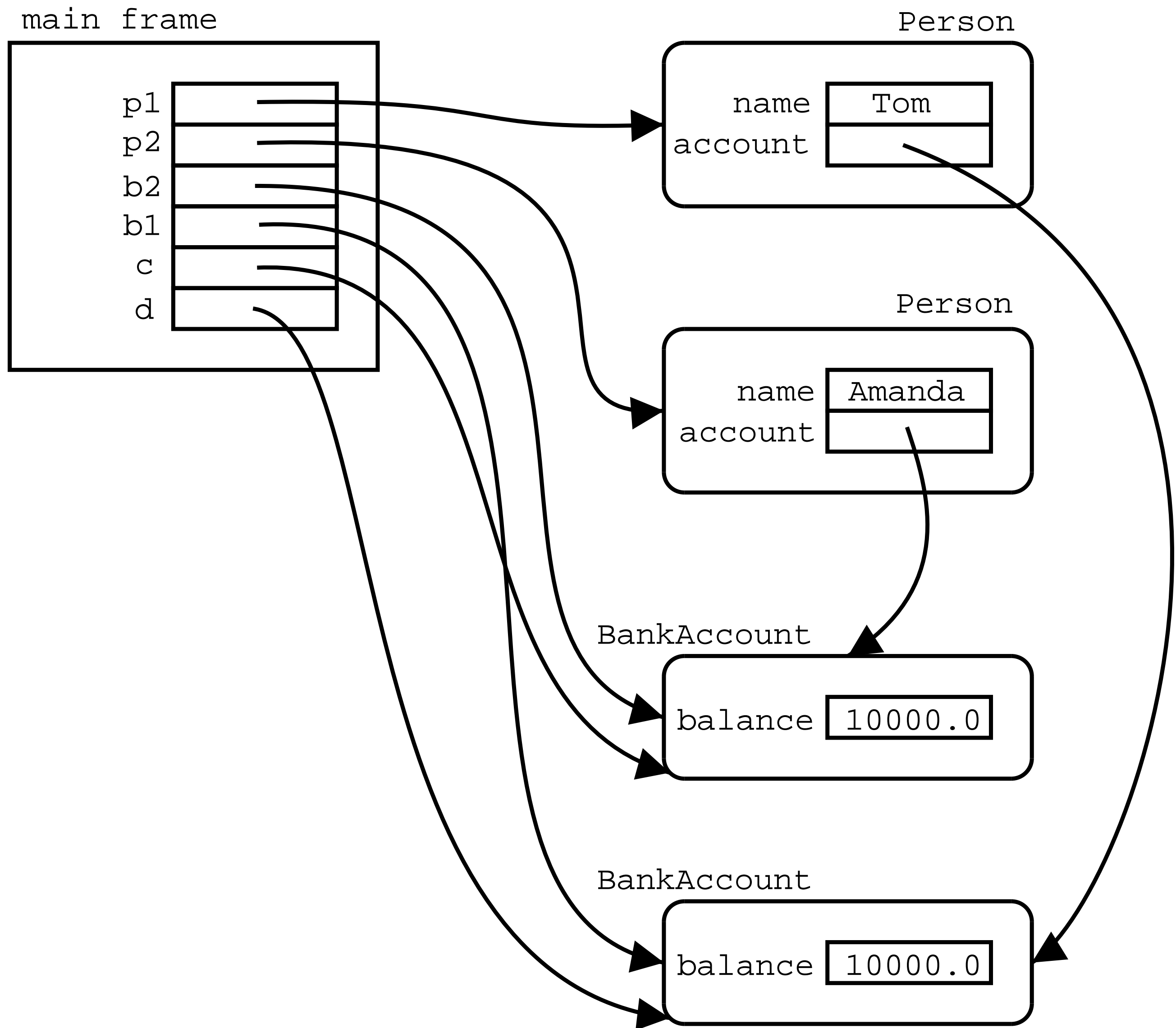
```
public class BankingTest
{
  public static void main(String[] args)
  {
    Person p1 = new Person(“Tom”);
    Person p2 = new Person(“Amanda”);
    BankAccount b1 = new BankAccount(10500.0f);
    BankAccount b2 = new BankAccount(10000.0f);
    p1.open_account(b1);
    p2.open_account(b2);
    BankAccount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

Example



McGill

Example



The end