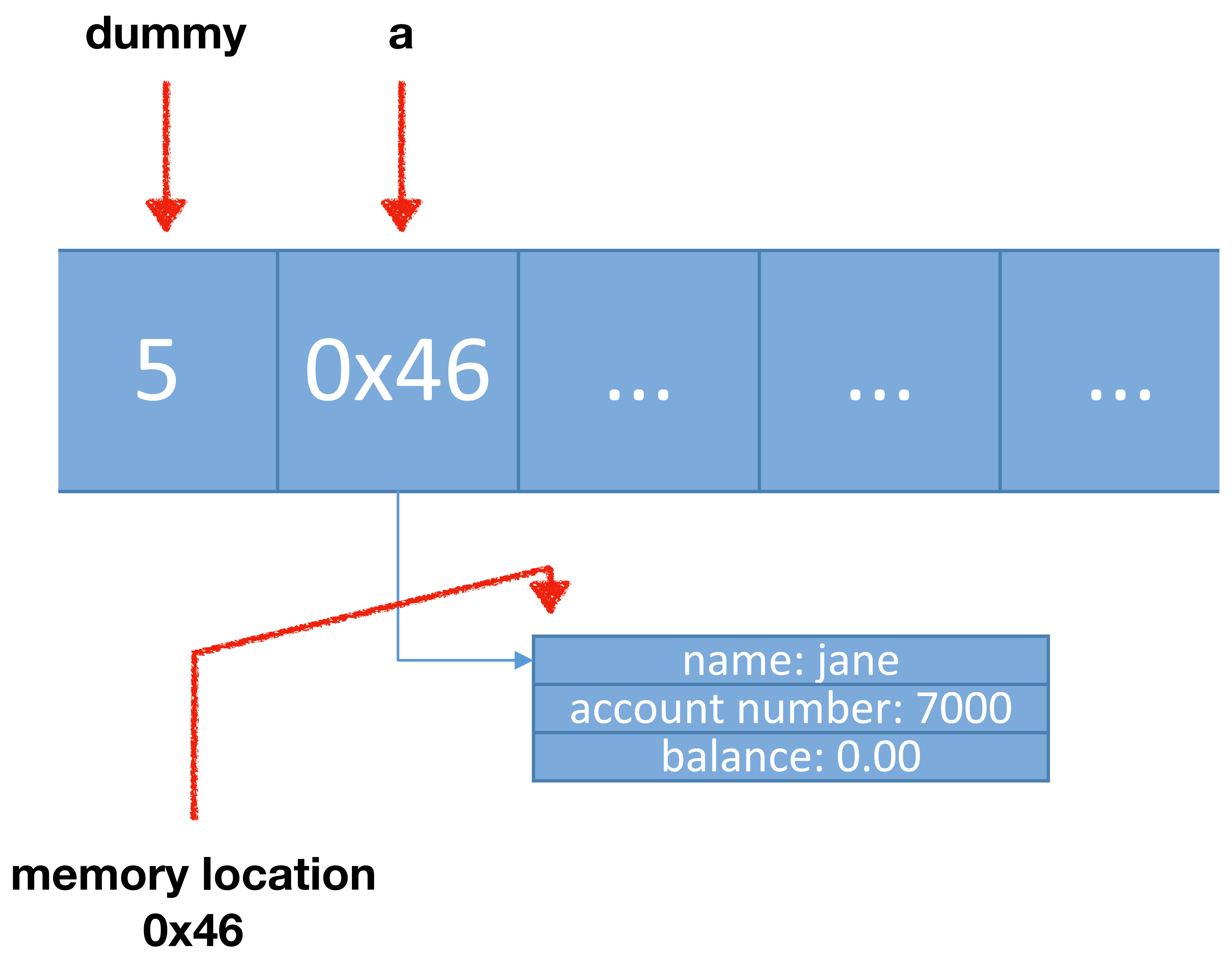
Programming Using Java

Session 4: More on Objects and Classes

```
public class Account {

  private String name;
  private int accountNumber;
  private double balance;

  public Account(String name, int number) {
    this.name = name;
    this.accountNumber = number;
    this.balance = 0;
  } }
```

```
public class BankApp {
  public static void main(String[] args) {
    double dummy = 5;
    Account a = new Account(“jane”, 7000);
    …
  } }
```



Reference Types

“…variables store primitive types, or references to objects…”

```
public class Account {

  private String openAccountName;
  private String name;
  private int accountNumber;
  private double balance;

  public Account(String name, int number) {
    this.name = name;
    if (number > 0) this.accountNumber = number;
    this.balance = 0;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getName () { return name; }
  public int getNumber () { return accountNumber;}
  public String getBalance () { return "$" +
                          this.balance)};
}
```

```
public class BankApp {

  public static void main(String[] args) {

    // create object of Account type
    Account a = new Account("jane", 7000);

    // change name
    a.openAccountName = "Open";
    a.name = "Closed";
    a.setName("joe");

    // print balance
    System.out.printf("Bal: %s", a.getBalance());

    // invalid account
    Account a2 = new Account("jim", -7000);
  }

}
```

Access Modifiers

"…allow control over how users can modify object's contents…"

```
public enum Month {

  // constant object declaration
  Jan(1, “January”), Feb(2, “February”), Mar(3, “March),Apr(4, “April”), May(5, “May”), Jul(7, “July”),
  Aug(8, “August”), Sep(9, “September”), Oct(10, “October”), Nov(11,”November”), Dec(12, “December”);

  // enum variables
  private final int monthNum; private final String monthStr;

  Month(int i, String s) {  monthNum = i; monthStr = s; }
  public int getMonthNum() { return monthNum; }
  public String getMonthStr() {return monthStr;}
}
```

enum constants

```
public class MonthTest {

  public static void main(String[] args) {
    System.out.println("Current Month:" + Month.Sep.getMonthStr());
    }

}
```

Enum Type

“…a fixed, unmodifiable set of objects…”

```java
public class Account {
  private String name;
  private int accountNumber;
  private double balance;
  private MyDate accountOpenDate;

  public Account(String name, int number) {
    this name = name;
    if (number > 0) this.accountNumber = number;
    this.balance = 0;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getName () { return name; }
  public int getNumber () { return accountNumber;}
  public String getBalance () { return “$” +
                      this.balance)};
}
```

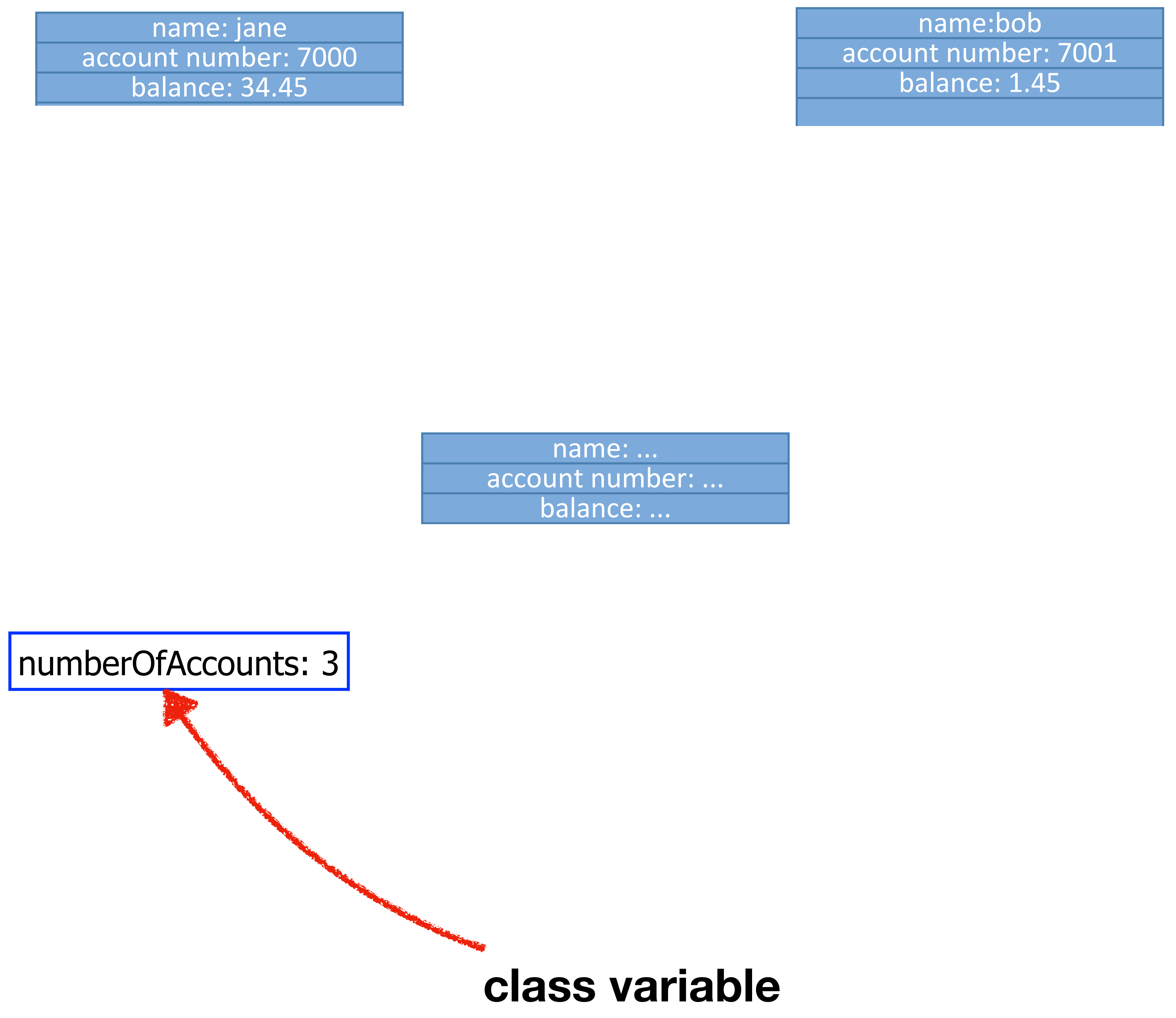
```java
public class MyDate {
    int day;
    Month mon;
    int year;

    MyDate (int d, Month m, int y) {
        if ((d > 0) && (d < 32))
            day = d;

        if (y > 0) year = y;
    }

    String getDate() {
        return mon.getMonthStr() + " " + day + ",
" + year;
    }

}
```

Composition

“…class definitions can have object references as instance variables…”

Static Class Members

```
public class Account {

   private String name;
   private int accountNumber;
   private double balance;
   private static int numberOfAccounts;
   …

   public static int getCount() {
     this.name = name;
   }
}
```



- class variables for all objects of a class type
- useful for tracking characteristics of a class
- static methods modify these variables

Inheritance

Classifying Objects

- classes are used to model real-world things or entities
- types can be organized into hierarchies
- “is-a” relationship basis
- allows objects to be an instance of many types simultaneously

Specialization

Within the “is a” relationship, there are two relative concepts

- superclass (or parent class)
  - all its instance variables and methods are in child
  - is a generalized version of children
- subclass (or child class)
  - possibly contains extra instance variables
  - methods shared with parent may behave differently (overriding)
  - considered a specialization of a class

Inheritance

- a subclass “inherits” instance variables and methods from its parent
- eliminates code duplication
- <u>super keyword</u> used to access parent methods
  - possibly contains extra instance variables
  - methods shared with parent may behave differently (overriding)
  - considered a specialization of a class

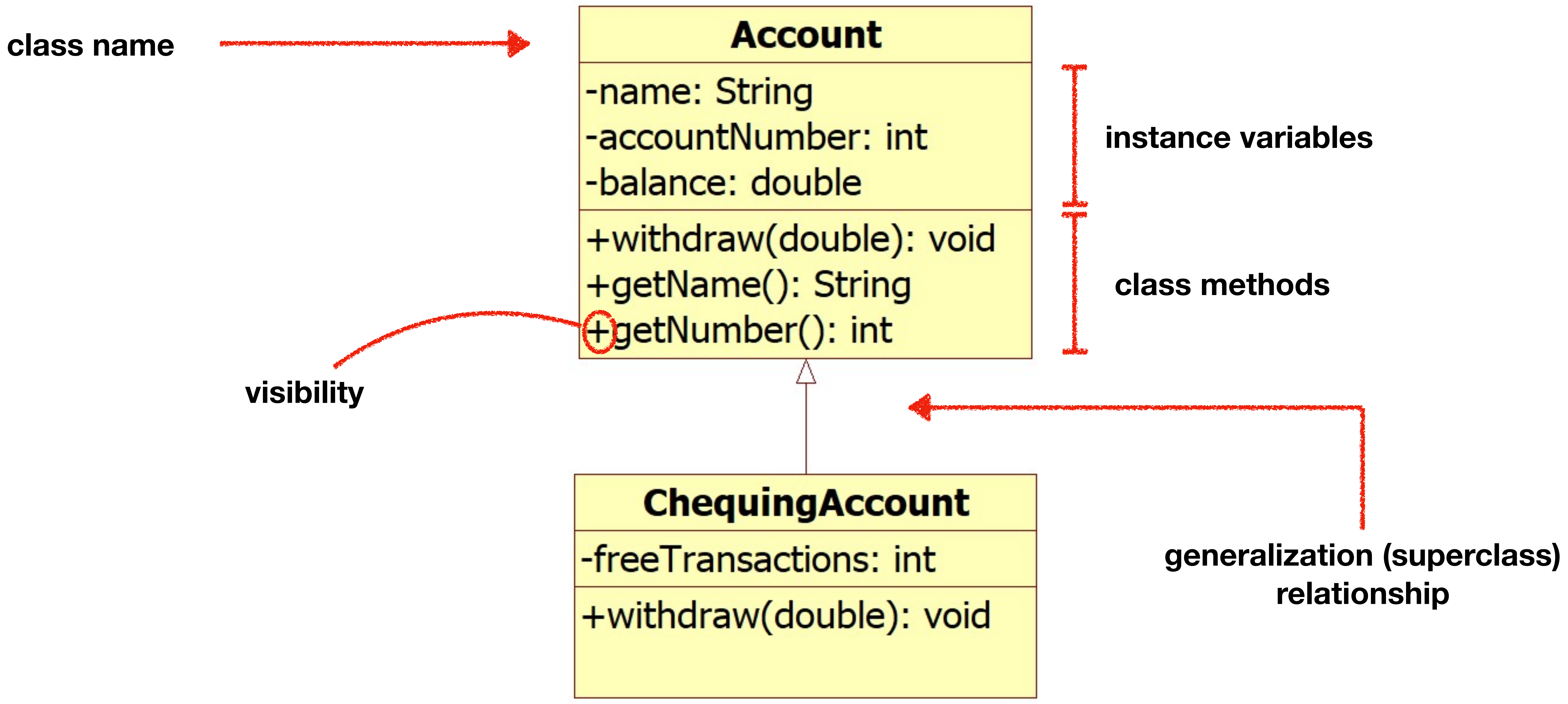
```
public class Account {
  private String name;
  private int accountNumber;
  private double balance;
  …
  public Account(String name, int number) {
    this name = name;
    if (number > 0) this.accountNumber = number;
    this.balance = 0;
  }

  public void withdraw(double w) {
    this.balance -= w;
  }

  public String getName () { return name; }
  public int getNumber () { return accountNumber;}
  public String getBalance () { return “$” +
                    this.balance)};
}
```

```
public class ChequingAccount extends Account {
  private static double transactionFee = 1.5;
  private int freeTranactions;

  …
  public ChequingAccount(String name, int number){
    super(name, number);
    freeTransactions = 5;
  }

  @Override
  public void withdraw(double w) {
    super.withdraw(w);
    if (freeTransactions < 1)
       super.withdraw(transactionFee);
    freeTranactions—;
  }
}
```

call parent’s version of withdraw

Superclass Implementation

“…child inherits parent’s instance variables and methods…”



UML Class Diagrams

"… documents structure of objects and relationships between object types …"