Introduction to Object-Oriented Programming

Christine T. Casey

VM/VSE Technical Conference

Session M60

May/June 2000

RETURN TO INDEX

Trademarks / Disclaimer

The following are trademarks of the International Business Machines Corporation:

VM/ESA
MVS/ESA
OS/2
AIX
OpenEdition
OS/390
S/390

The following terms are trademarks or registered trademarks of other companies or institutions:

Java (Sun MicroSystems, Inc.)
Windows (Microsoft Corporation)

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "as is" basis without any warranty either expressed or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environment do so at their own risk.

In this document, any references made to an IBM licensed program are not intended to state or imply that only IBM's licensed program may be used; any functionally equivalent program may be used instead.

Any performance data contained in this document was determined in a controlled environment and, therefore, the results which may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environments.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Agenda

- Principles of OO Programming
  - Classes, Objects, Methods
  - Encapsulation
  - Inheritance
  - Composition
  - Polymorphism
  - Reuse
- Potential Benefits and Concerns of OO
- Summary of concepts

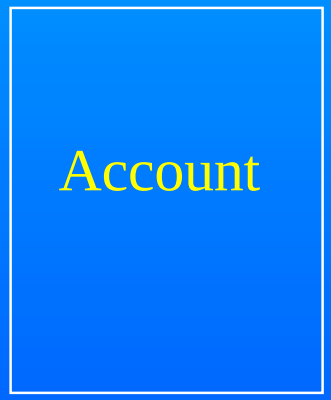
Classes and Objects and Methods... (oh my!)

- Class - template used to create objects
  - the cookie cutter
- Object - the space allocation
  - the cookie
- Class Characteristics:
  - Combine data and behavior in one package
    - Protective of their hidden data
    - Known as encapsulation
  - Described by nouns
  - Contain Methods (the verbs) providing functions and procedures

Object Concepts - Methods

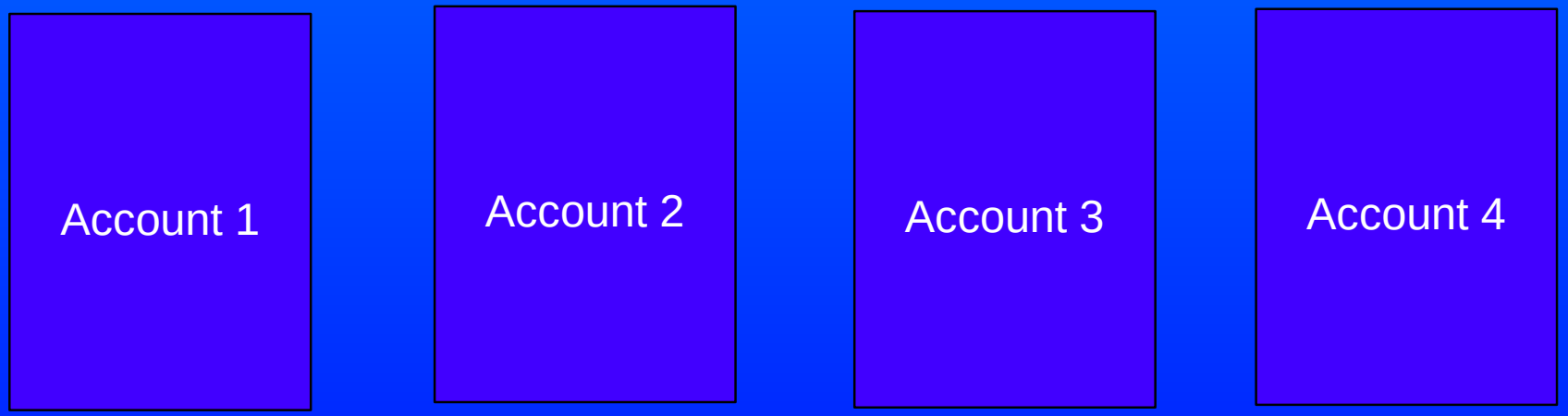
- Constructors
  - Runs when object is instantiated
  - Same name as object
  - Initializes object's data
- Overloading - write_check(int)
  write_check(int,int)
  - Same method, different arguments
  - Method with args matching the call is run
- Overriding - deposit(int)
  - Same method name in different objects
  - Used by child object to replace a parent's method

Object or Class?

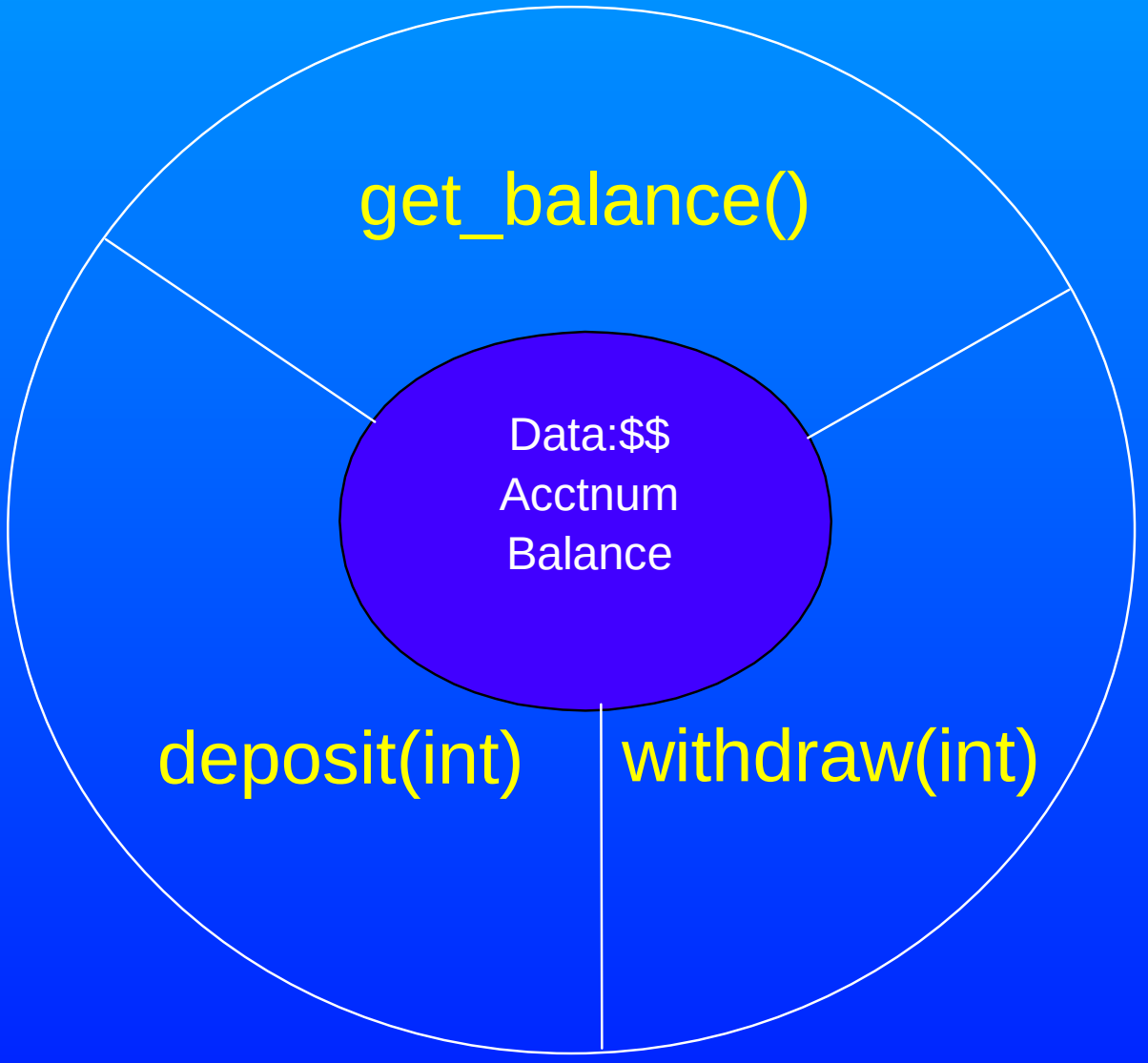


Class (use as template)

Objects
(instantiated from class)



Object Concept - Encapsulation



packaging of related data and procedures together

Class, Object, and Method Example in Java

```
class  Account  {

        int Balance;                // object data
        void Account( )  {          //  constructor
            Balance = 0;            //  initialize
    }
        void deposit(int  i)  {     //  methods
            Balance += i;
    }
        void withdraw(int  i)  {
            Balance  -= i;
    }
        void get_balance( )  {
            return Balance;
    }
```

Class, Object, and Method Example in Java (cont.)

```
public static void main(String argv[ ] )
 {
  //  create new objects and call methods

   Account account1 = new Account( );
   Account account2 = new Account( );

   account1.deposit(25);
   account2.deposit(100);
   account1.withdraw(5);

System.out.println("balance="+account1.get_balance());
   }
 }
```

Method Example in NetRexx

```
/* NetRexx Method Example */

A = Account(  )              -- set up an account object
A.deposit(int 25)            -- call deposit method
A.deposit(int 100)           -- call deposit to add more money
A.withdraw(int 5)            -- call withdraw method
new_Balance = A.Get_Balance(  )   -- call Get_Balance
say "Balance =" new_Balance       -- print new balance
```

Method Example in NetRexx

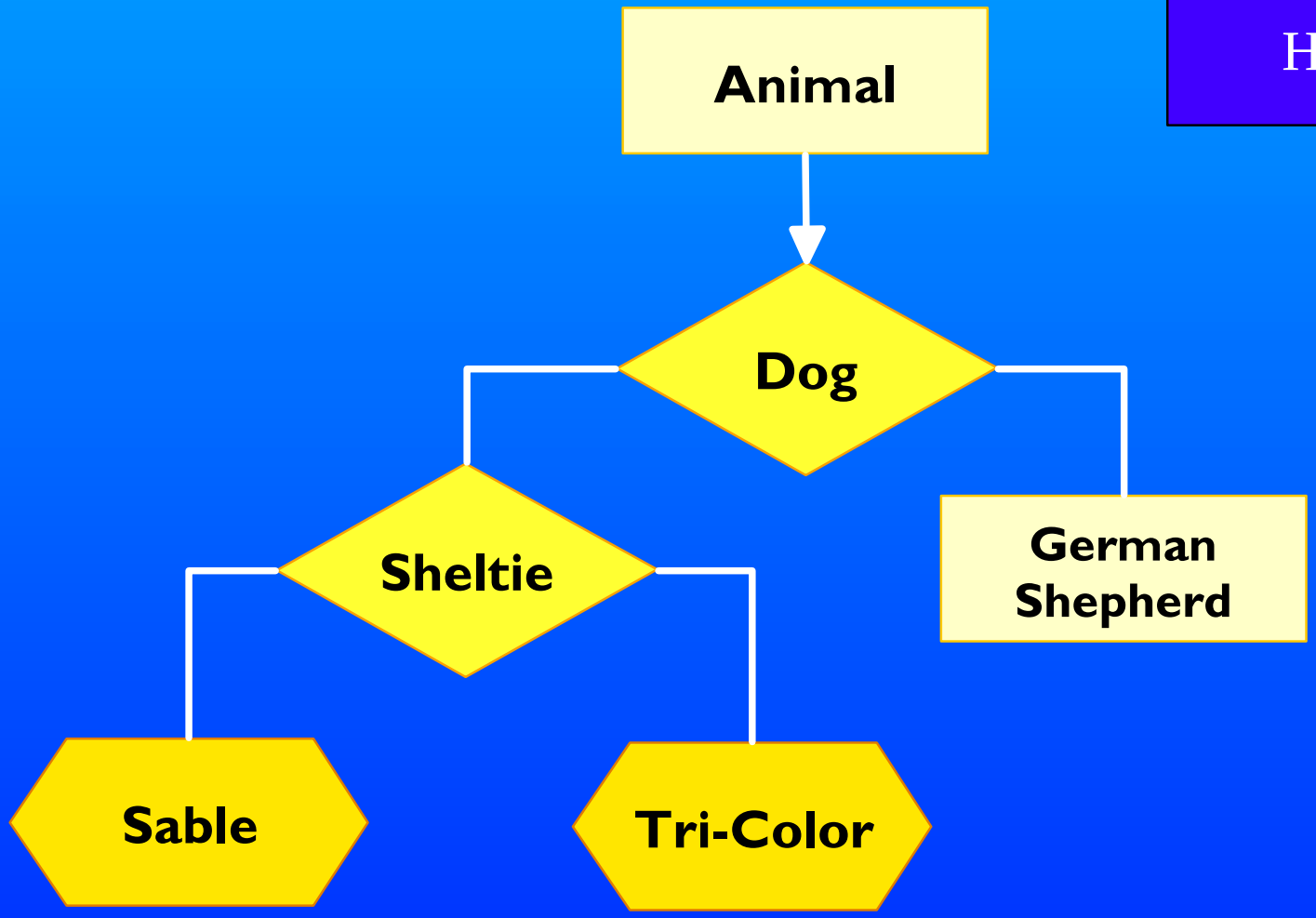
```
class Account
   Balance = int                    -- property  type integer

   method Account(  )               -- constructor
      Balance = 0

   method deposit(in_money)
      Balance = Balance + in_money

   method withdraw(out_money)
      Balance = Balance - out_money

   method Get_Balance returns int
      return Balance
```

Inheritance

- Used in implementing OO-relationships
  - sending messages between objects
  - create objects as part of a new object
- Kind-of (Is A)
  - Dog is a "Kind-of" animal
  - "Is a" inheritance to reuse by extending other classes
- Part-of (Has A)
  - Tail is a "Part-of" dog (Dog "has a" Tail)
  - "Has a" composition to reuse by including other classes

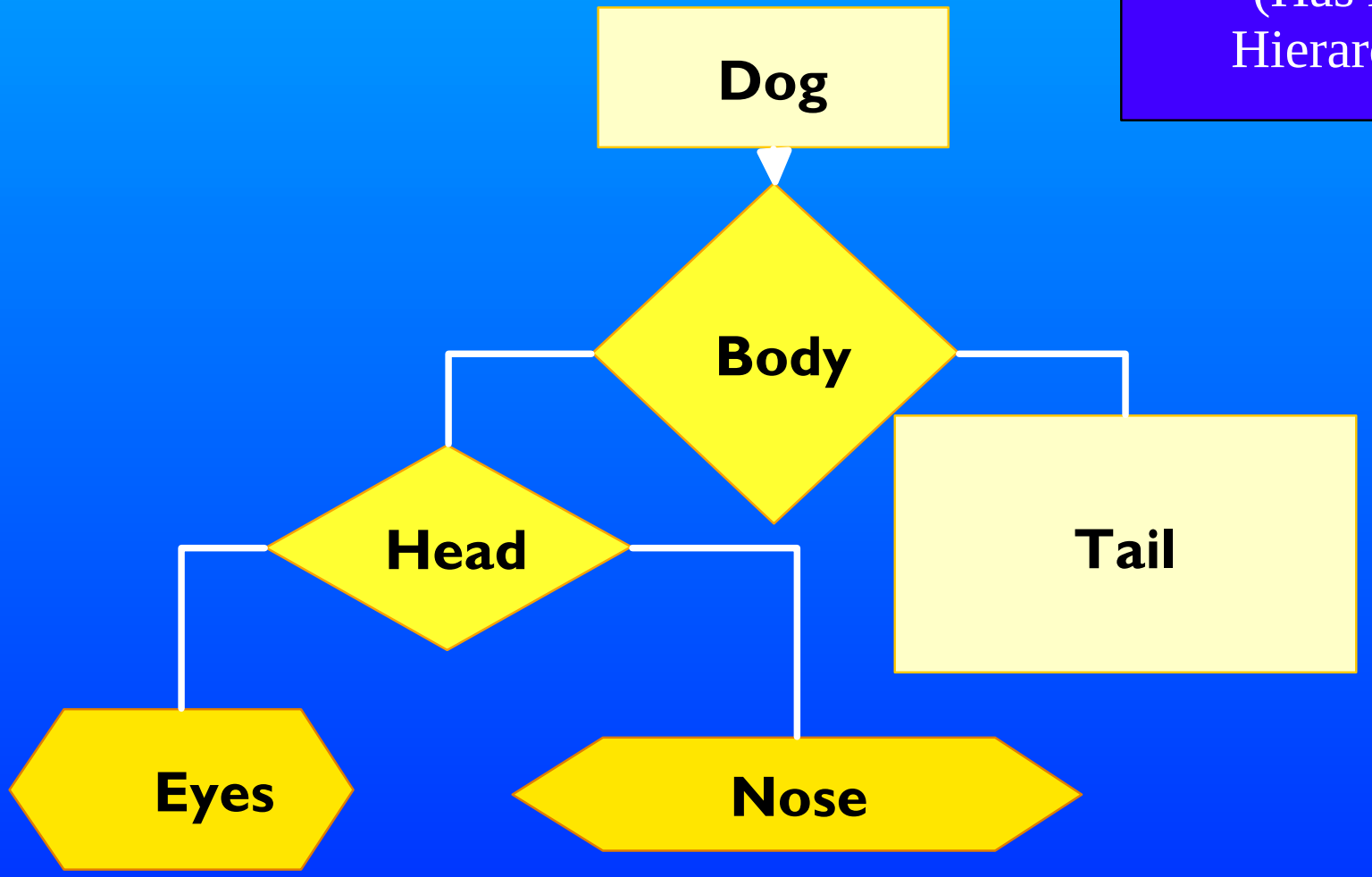
Inheritance

Example of Kind-Of (Is A) Hierarchy



Inheritance

Example of Part-Of (Has A) Hierarchy



Composition

- Build class consisting of other classes
- Instances are composite objects
- Related to "Has A" concept

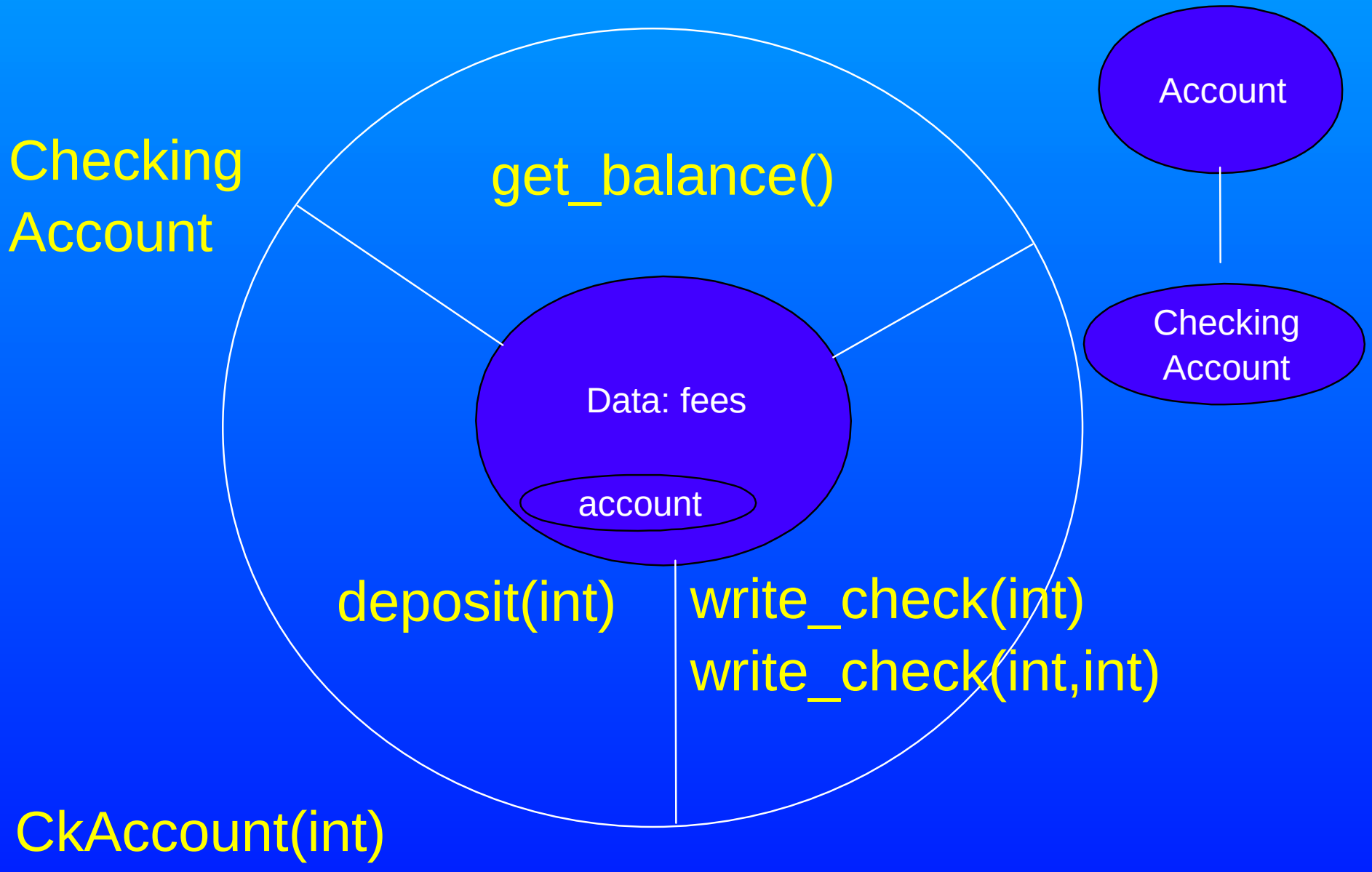
Inheritance and Composition Example: Is a, Has a

Java class description in English:

"A home **is a** house which **has a** family and a pet"

```
public class Home extends House  {
        Family  humans;              // composition
        Pet  animal;
}

//  extend by inheriting (is a)   or
//  compose by building blocks  (has a)
```

Object Concept - Inheritance



Code Example: Inheritance, Overloading

```
class CkAccount extends Account  {

  int  fees;
  CkAccount(int  f)  {
      fees = f;                        //  set fee passed in
   }
  void write_check(int amount)  {
      withdraw(amount);
   }
  void write_check(int amount,int f)  {
       withdraw(amount);
       System.out.println("We need the fees");
  }
}
```

Code Example: Inheritance, Overloading (cont.)

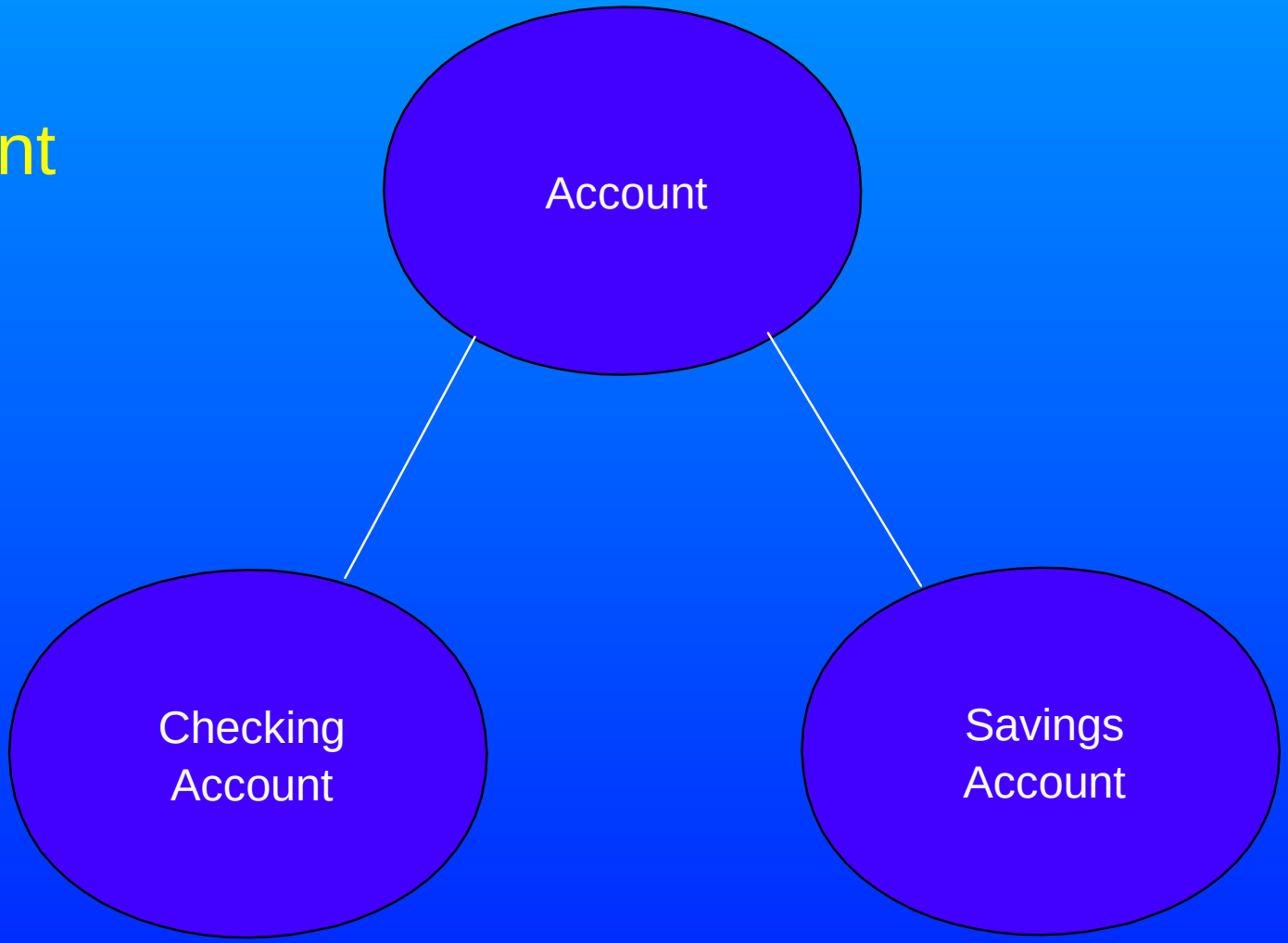
```
public static void main (String argv[ ] )  {
  CkAccount ck1 = new CkAccount(1);
  System.out.println("Bal 0 ="+ck1.get_balance( ));
  ck1.deposit(100);
  System.out.println("Bal 1 ="+ck1.get_balance( ) );
  ck1.write_check(10);
  System.out.println("Bal 2 ="+ck1.get_balance( ) );
  ck1.write_check(20,0);
  System.out.println("Bal 3 ="+ck1.get_balance( ) );
   }
}
```

Polymorphism

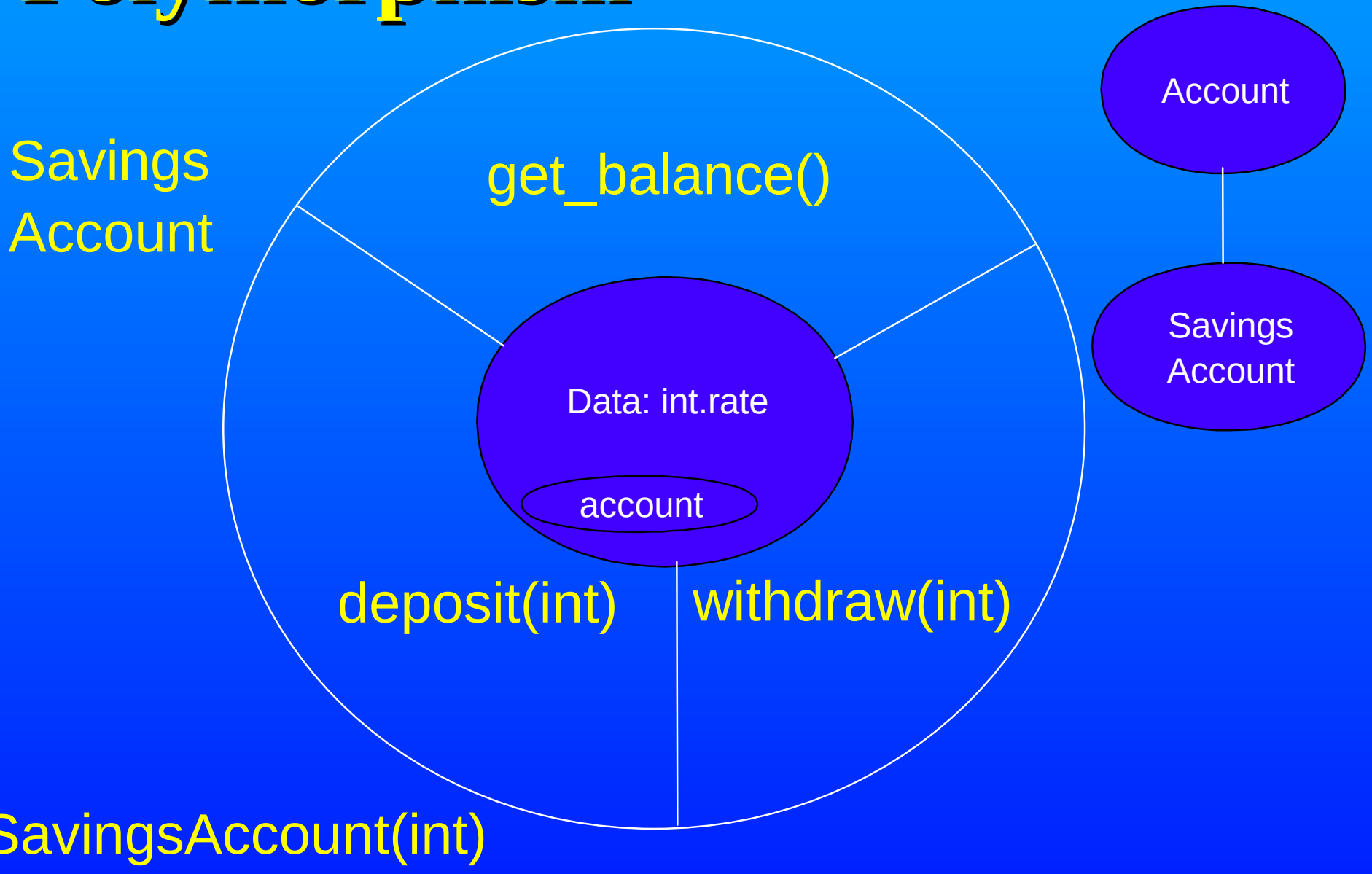
- Capability of a single variable to refer to different objects
  - Account can hold SavingsAccount or CheckingAccount object at different times
- This allows a new object to be added without rewriting existing procedures

Object Concept - Polymorphism

Account



Object Concept - Polymorphism



Reuse

- Once a class is defined, all instances of that class are guaranteed to be identical and perfectly formed
- Same name methods in unrelated class and unrelated methods
- Reusable objects are building blocks for future software
- Create classes complete enough for expected needs, but simple and independent of other classes

Potential Benefits of OO Programming

- Faster Development
- Higher Quality of Code
- Easier Maintenance
- Reduced Cost
- Increased Scalability
- Better Information Structures
- Increased Adaptability

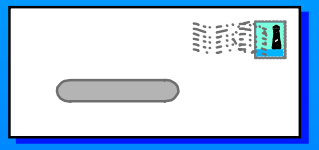
Potential Concerns of OO Programming

- Maturity of the Technology
- Need for Standards
- Need for Better Tools
- Speed of Execution
- Availability of Skilled people
- Costs of Conversion
- Support for Large-Scale Modularity

Summary

- Different way to design reusable software
- Keep it Simple:
  - choose meaningful variable, method, and class names
  - nouns for classes, verbs for methods
  - avoid large complex classes in favor of several smaller ones
  - data should be private with reasonable number of access methods
- You can do it !!

Contact Information



Christine Casey
ccasey@vnet.ibm.com
caseyct@gdlvm7

Fax: (607)752-1162
Phone: (607)752-5049