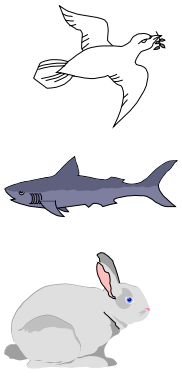
Dynamic Binding and Polymorphism

- **Polymorphism and Dynamic Binding**
- **Casting and Conversion**
- **Use of keyword instanceof**
- **Access Control (public, protected, private, default)**
- **Class (static) methods and variables**
- **final methods classes and data**
- **Procedural programming**

1

Polymorphism

- **When I tell someone at home to feed the pets they know it means different things for different type of pets.**
- **Hence We can consider the action feed to be polymorphic.**
- **In OOP, polymorphism promotes code reuse by calling the method in a generic way.**
- **For example we can say deduct $5.00 monthly charges by calling the withdraw() method on all account objects. But depending on the type of account the correct version of withdraw() will be called.**



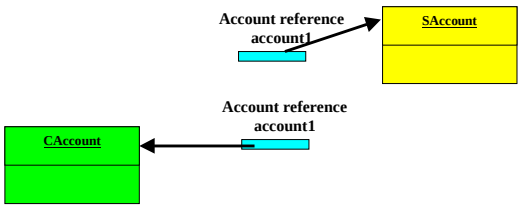
2

Object Reference Conversion

To enable methods to be called in a polymorphic way Java allows a superclass reference to refer to a subclass object.

As SAccount and CAccount extend the class Account both of these statements are valid.

```
Account account1;
. . .
account1  = new SAccount("s123","Tom", 100, 0);
. . .
account1=new CAccount("c123","Kim",2000,1000);
. . .
```

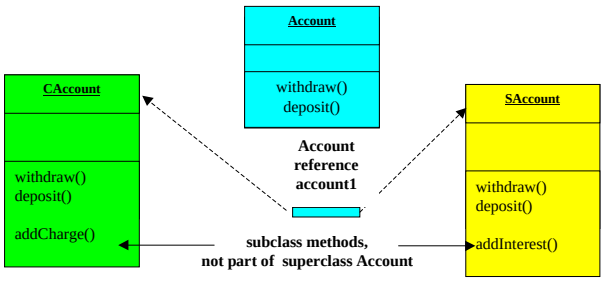


3

Invoking methods through Superclass Reference

When `Account` reference `account1` referring to a *subclass* object is used to invoke a method such as withdraw(), the *overridden* `withdraw()` of that *subclass* is invoked.

```
Account account1 = new SAccount("s123","Tom",100, 0);
account1.withdraw(100); // withdraw of SAccount invoked
account1 = new CAccount("c343","Kim",2000,1000);
account1.withdraw(200); // withdraw of CAccount invoked
```



4

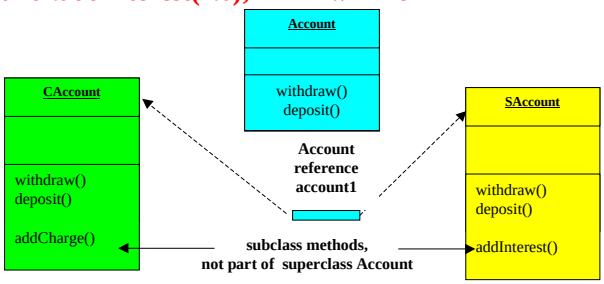
Invoking methods through Superclass Reference

However, the compiler will report an error if we attempt to call the addInterest() of SAccount or deductCharge() of CAccount through an Account reference.

Can I say walk the pets if I have a fish, dog and a bird as a pets ?

Account account1 = new SAccount("s123","Tom",100, 0);

account1.addInterest(1.0); // Error



5

Summary

- **A *superclass* reference can refer to a *subclass* object.**
- **However only methods of that *superclass interface* can be called through this *reference*. The actual method called (whether it is of the superclass or one of its subclasses) depends on the type of object being referred.**
- **To call other methods not found in the superclass we need to *cast* the *reference* to the appropriate type.**

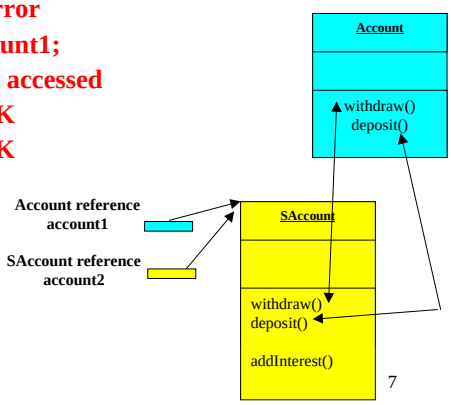
6

Casting

If we know that the Account reference account1 is referring to a SAccount object, we can use a cast to convert it back. Such a reference can then be used to call any of the subclass methods.

```
Account account1 = new SAccount("s123","Tom",100, 0);
account1.withdraw(100);          // OK
account1.addInterest(1.0);       // Error
SAccount account2 = (SAccount) account1;
// now all methods of SAccount can be accessed
account2.withdraw(100);          // OK
account2.addInterest(1.0);       // OK
```

If however the reference `account1` happened to point to an object of some other type (other than `SAccount`) an exception will be thrown at run-time.



7

Summary of Casting

- **A superclass reference can be cast to a subclass reference if we are sure it is referring to a subclass object.**
- **The program will throw an exception and terminate at run-time if it is cast wrongly.**

8

An analogy in English ...

We have a new tutor Mr Lim. He will be taking our tutes.

We can safely replace the the reference *he* by *that-person* to get

We have a new tutor Mr Lim. That person will be taking our tutes.

We have a new tutor Poon Yap Ching. That person will take all our tutes.

We can replace the the reference *that-person* by *he* if we know that person is a male.

We have a new tutor Mr Lim. He will be taking our tutes.

We cannot however replace the reference e *he* by *she*

9

Quiz

What will be the outcome?

```
Account a1 = new Account("s123","Tom",100);
SAccount a2 = (SAccount) a1;
```

a) Compilation error

b) Program will terminate at runtime with ClassCastException

c) No error

answer is (b), as a1 was referring to an Account object

10

Quiz

What will be the outcome?

```
CAccount a1 = new CAccount("s222","Jack",500);
SAccount a2 = (SAccount) a1;
```

a) Compilation error

b) Program will terminate at runtime with ClassCastException

c) No error

answer is (a) as CAccount is not a subclass of SAccount, hence the compiler will report an error. They are unrelated classes.

11

The Operator `instanceof`

End of month processing requires us to invoke `addInterest()` to all `SAccount` objects and `deductCharges()` to all `CAccount` objects.

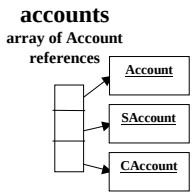
The method `monthEndProcessing()` below is passed an `Account` (superclass) reference. This method uses `instanceof` to determine the type of object being referred.

```
void monthEndProcessing (Account a)  {
      if (a instanceof SAccount)
            ((SAccount) a).addInterest(1.0);
      else if (a instanceof CAccount)
            ((CAccount) a).deductCharges();
}
```

12

Use of Polymorphism

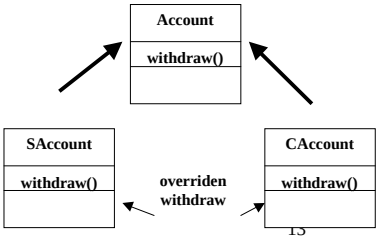
```
Account[] accounts = new Account[3];

accounts[0] = new Account("a12345","Charles",1000);
accounts[1] = new SAccount("s12346","Craig",1200,1000);
accounts[2] = new CAccount("c12347","George",200,1000);

// Deduct fixed amount $500 from all accounts
for (int i=0; i<3; i++)
        accounts[i].withdraw(500);
```



Which withdraw() methods are called ?

Though Accounts[i] is a Account reference in Java, actual method called is determined at run-time based on the type of object being referred.

This feature is Called Polymorphism.

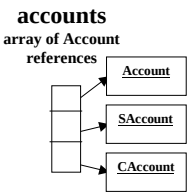


13

What if the language offers no support for polymorphism (dynamic binding) ?

Assume Account class has a method getType() returning the type of account. Then the code to withdraw $500 from all accounts:

```
// Deduct fixed amount $500 from all accounts
for (int i=0; i<3; i++) {
    if (accounts[i].getType() == "Account")
            accounts[i].withdraw(500));
    else if (accounts[i].getType() == "SAccount")
            ( (SAccount) accounts[i]) .withdraw(500));
    else if (accounts[i].getType() == "CAccount")
            ( (CAccount) accounts[i]) .withdraw(500));
}
```



If a new type of account (Fixed Deposit) is added do we need to change the code ? Yes, add an extra else if statement.

With a Language that supports polymorphism the code need not be changed !!! Polymorphism helps to promotes Code Reuse.

14

Another Example of Polymorphism

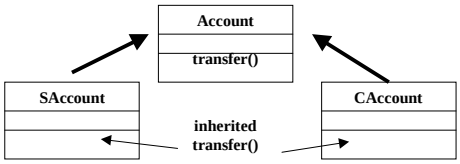
To transfer $100 from the first account to second, second to third and so on...

```
for (int i=0;i<2; i++)
  accounts[i].transfer(accounts[i+1],100);
```

Same as writing:

```
accounts[0].transfer(accounts[1],100);
accounts[1].transfer(accounts[2],100);
```

But unlike withdraw(), transfer() is a superclass method not overridden in the subclasses.



15

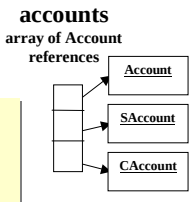
Another Example of Polymorphism contd'

Definition of transfer in Account class uses deposit() and withdraw() methods

```
public boolean transfer(Account account, double amount) {
   if (this.withdraw(amount)) { // same as withdraw()
        account.deposit(balance);
        return true;
   }
   else return false;
}
```

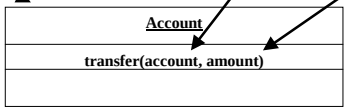
Which withdraw() ?

Which deposit() ?



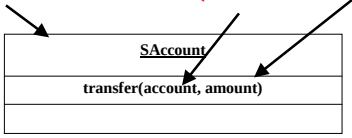
Thanks to late-binding, the actual method invoked depend on the object they are acting upon (refer to the array).

accounts[0].transfer(accounts[1],100);



Calls withdraw() of Account and deposit() of SAccount

accounts[1].transfer(accounts[2],100);

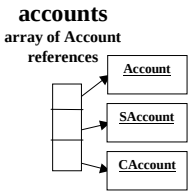


Calls withdraw() of SAccount and deposit() of CAccount

16

Calling Subclass Methods through Superclass references

- Suppose we want to add interest to all the SAccount objects and deduct monthly charges from all CAccount objects using the array of Account (superclass) references.
- The code below first uses the instanceof keyword to determine the type of object being referred.
- Before calling the subclass method the superclass reference is cast to the appropriate type.




```
// In the month end, add interest to all saving accounts
// and deduct charges from all checking accounts
for (int i=0;i<3; i++) {
   if (accounts[i] instanceof SAccount)
       ((SAccount) accounts[i]).addInterest(1.0);
  else if (accounts[i] instanceof CAccount)
       ((CAccount)accounts[i]).deductCharges();
}
```

Method of SAccount

Method of CAccount

17

Packages

- In Java package is the equivalent to library in other languages
- We can seen how to make all the classes from a particular package available with the statement like `import java.io.*`
- We can also place our own classes is a specific package by adding a statement like `package myPackage;` at the top of each source file.
- If no package name is provided Java takes the current directory as the package directory

18

Summary of Access Control

Java has four levels of controlling access to variables, methods and classes:

- **public**
- **private**
- **protected access**
- **package access**

Modifier	*Class visibility*	*Methods and variables visibility*
`public`	Everywhere	Inherited by all subclasses Accessible anywhere
`Default`	Package	Inherited by subclasses in the same package Accessible by any class in the same package
`Protected`		Inherited by all subclasses Accessible by any class in the same package and inherited classes
`Private`		Inherited by all subclasses Not accessible

19

Inherited Methods and Accessibility

A subclass cannot override a method to become less accessible. For example, the compiler will not allow this.

```
class Account {
        public void withdraw(double amount) { . . . }
                . . .
}
class SAccount extends Account {
        protected void withdraw(double amount) { . . . }
        . . .
}
```

20

Class (Static) Variables and Methods

In our `SAccount` class, we had a method for adding monthly interest which takes the rate as argument.

```
void addInterest(double rate);
```

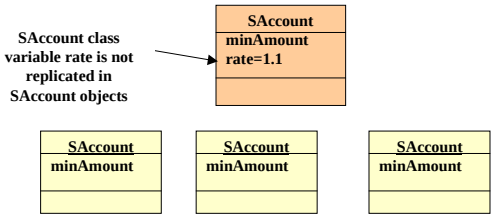
What if all saving account holders are paid same rate?

We could make the rate a class variable, one that is common to all instances.

The keyword `static` indicates a class variable.

```
class SAccount {
  …
  // instance variable (varies for different account holders)
  private double minAmount;
  // class variable rate (common to all account holders) set to 1.1
  public static double rate = 1.1;
```

21



Now the addInterest() method can be overloaded to use the class variable rate as in:

```
public void addInterest(double r)  {
  // using the class variable rate
  deposit(getBalance() * r / 100);
}
public void addInterest()  {
  // using the class variable rate
  deposit(getBalance() * rate/100);
}
```

Original method

Overloaded method using the class variable rate

22

Can we reset the static variable ?

In the test (driver) class we can reset the `static` variable `rate` if necessary and use the subclass `addInterset()` as in:

```
SAccount.rate = 1.2;
    for (int i=0;i<3; i++)
        if (accounts[i] instanceof SAccount)
            ((SAccount) accounts[i]).addInterest();
```

23

Class (static) methods

If however, we make the class variable rate private (we should) then we could allow them access through class methods using the static keyword as in:

```
public static void setRate(double rate)     {
       SAccount.rate = rate;
}
public static double getRate() {
       return SAccount.rate;
}
```

Note: class name used

Then the driver class can be coded as in:

```
SAccount.setRate(1.2);
for (int i=0;i<3; i++)
   if (accounts[i] instanceof SAccount)
      ((SAccount) accounts[i]).addInterest();
```

24

Restriction on class methods

- **Some class methods we have used before include `parseInt()` and `sqrt()`.**

```
int num = Integer.parseInt(s);
double d = Math.sqrt(9.5);
```

- **We were able to use those class methods without having to create an instance of that class.**
- **This means that the class method cannot access any instance variables or methods as they will not come into existence until an object is created explicitly (with keyword new).**

25

Exercise

Identify the errors in the class method `getTotalInterest()` trying to compute the total interest earned over a period of *n* months given an initial amount and interest rate.

```
class SAccount extends Account {
  …
  public static double getTotalInterest(double
        initialAmount, double rate, int n)
  {
        // balance - instance variable of superclass Account
        balance = initialAmount;
        // instance method
        for (int i=0; i<n; i++)
              addInterest(rate);
        return balance – initialAmount;
  }
```

26

Final Primitives and References

- We have seen `final` used to state something is constant.
- The `final` different meanings depending on where it is used, but in general it means "this cannot be changed."
- It can be used for data, methods and for a class itself.

```
// final primitive
final int MAX_STUDENTS = 100;
// final reference (object can be changed)
final Employee currentEmp = new Employee();
```

27

Final methods and classes

- For methods, `final` prevents the method from being overridden by any inheriting class.
- Ensures that the behaviour of a method is retained during inheritance and cannot be overridden.
- For classes `final` means it cannot be extended.
- When a class is declared `final` that class cannot be subclassed, and of course its methods cannot be overridden because the class itself cannot be extended.
- The class `String` is defined:

```
public final class String
```

Therefore it cannot be extended.

28

Procedural programming

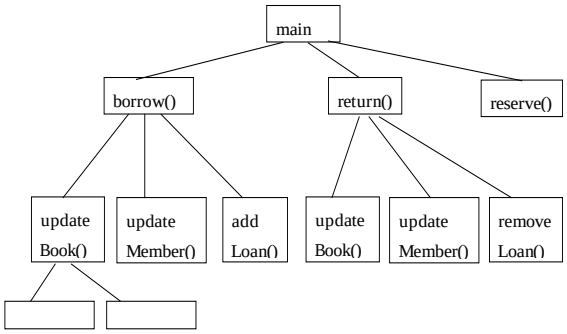
- **In *procedural programming* the code is modularized based on system's processes.**
- **C Programmers concentrate on writing functions.**
- **Though data is important the view is that data exists primarily in support of the actions that functions perform.**
- **For instance, a library application system would have processing involving borrowing, returning, reserving and cataloging of books etc.**

29

```
struct book {
       char ID[10];
       char title[30];
       char author[30]
       char status;
};
struct Member {
       char ID[10];
       char name[30]
       int booksBorrowed;
};
int borrow (int  bookID, int memberID) {
       …
}
int return (int  bookID, int memberID) {
       …
}
int reserve (int  bookID, int memberID, date D) {
       …
}
…
```

30

Structure Chart Showing the Procedural Flow



z

31