**Inheritance**

- idea is to build on a more general class, by extending it to do more specialized stuff for a subset of the members of the class
- provides a way to reuse the common elements of the superclass
- for example, *GeometricObjects* could be the more general class (superclass), and *Circle* is the specialized subclass, as in the text.
- terminology is: *Circle extends GeometricObject*

- another example: demo BankAccount class, and SavingAccount is special type of BankAccount that gives interest.

```java
public class BankAccount
    private double balance;

public class SavingsAccount extends BankAccount
    private double interestRate;
```

```
public class BankAccount
    superclass (also parent class, base class)

private double balance;

public class SavingsAccount extends BankAccount
    subclass (also child class, derived class)

private double interestRate;

```

automatically gets all BankAccount instance variables plus this
Inheritance Example

```java
// Illustrates BankAccount superclass.
/* @author berger */
/* @version 1 */
*/

public class BankAccount {
    private double balance;
    public BankAccount(double initialBalance){
        balance = initialBalance;
    }
    public BankAccount() { // default - no-arg constructor
        this(0); // must be 1st line of constructor to use 'this'
    }
    public double getBalance(){
        return balance;
    }
    public void deposit(double amount){
        balance += amount;
    }
    public void withdraw(double amount){
        if (amount > balance){
            System.out.println("Insufficient funds to withdraw": amount);
            System.out.println("Transaction denied");
        } else
            balance -= amount;
    }
    public String toString(){
        return "account has "+ balance + "+ dollars";
    }
    public void transfer(BankAccount other, double amount) {
        withdraw(amount);
        other.deposit(amount);
    }
}

/* Illustrate subclass which adds instance variables. */
/* @author berger */
/* */

public class CheckingAccount extends BankAccount{
    private int numTransactions;
    private static final int FREE_TRANSACTIONS = 3;
    private static final double TRANSACTION_FEE = .50;
    public CheckingAccount(double initialBalance){
        super(initialBalance);
        numTransactions = 0; // checking account has 3 free transactions
    }
    public CheckingAccount(){
        this(0);
    }
    /* this method overrides superclass methods, then calls it directly. */
    @Override
    public void deposit(double amount){
        numTransactions++;
        super.deposit(amount);
    }
    /* this method overrides superclass methods, then calls it directly. */
    @Override
    public void withdraw(double amount){
        numTransactions++;
        super.withdraw(amount);
    }
    public void deductFees(){
        if (numTransactions >= FREE_TRANSACTIONS){
            double fees = TRANSACTION_FEE * (numTransactions-FREE_TRANSACTIONS);
            super.withdraw(fees);
        }
        numTransactions = 0;
    }
    @Override
    public String toString(){
        return super.toString() + "; numTransactions "+ numTransactions;
    }
}
} // end CheckingAccount
```
Inheritance

• **super** keyword: invokes superclass constructor. must be first line in subclass constructor

• if superclass is itself a subclass of yet another class, it automatically invokes its superclass constructor, all the way up the chain (constructor chaining)

• **super()** calls the no-arg constructor, or super(params) call the ones the thatches the arguments.

• if you don’t explicitly call a superclass constructor, its default no-arg constructor is called for you. So there better be one, or its an error.

• **super** keyword also used to invoke superclass methods
  
  
  \[ \text{super.methodName(params)} \]

• run demo of InheritanceExample1.java
• **Overriding Methods:**
  
  • a subclass can use its superclass methods, since it IS a member of the parent class
  
  • if you want to replace the parent’s method with your own more specialized one you write a method with SAME signature AND return type in the derived class. (If it is a different signature it is **method overloading**).

• **demo of InheritanceExample2.java** overriding the `toString()` method

• a private method cannot be overridden since it is not accessible.

• private data in superclass not accessible (unless there are public getters).  
  
  *(protected data is accessible to subclasses)*

• to call static methods in parent class use `SuperClassName.staticMethodName()`
Inheritance

• every class is descended from the *Object* class.

• If no superclass specified with extends, then by default it extends *Object* but no need to say it explicitly.

• The Object class’ `toString()` method is called by default from `System.out.println()`, which prints name of class and memory address.
  
  ```java
  System.out.println(object)  equivalent to  System.out.println(object.toString())
  ```
Polymorphism: many shapes

• suppose have `class A`, and `class B extends A` e.g. BankAccount and CheckingAccount ex.
• can declare variables:

```java
BankAccount b;
CheckingAccount c;
```

and can pass `c` to a method expecting a `b`, because `c` is a type of BankAccount.

Since a subclass is a type of its superclass, a variable of a supertype can refer to a subtype.

• suppose call a method with a variable `v` that has a chain up to Object. There may be several methods with same name and signature in the chain. The decision of `which method to call` is made at runtime (i.e. dynamically) by the JVM, and the most specific method is used depending on actual type of object. Called `dynamic binding`
More on overriding: (.equals method, compareTo, etc.)

• the Object .equals methods tests the references:
  \[ o1 == o2 \]

• to override the .equals() method need to have same signature, which takes an Object

```java
public boolean equals (Object o){
    if (o instances MyTime){ // explicit test
        MyTime t = (MyTime) o; // explicit cast of o to new variable t
        return (hour == t.hour && minute==t.minute && second == t.second)
    }
    else  return (this == o); // use same definition as in Object class
}
```