

Yasir Ahmad
Visiting Faculty Member
From
ICS & IT Department
The University Of Agriculture Peshawar

cslearnerr.com

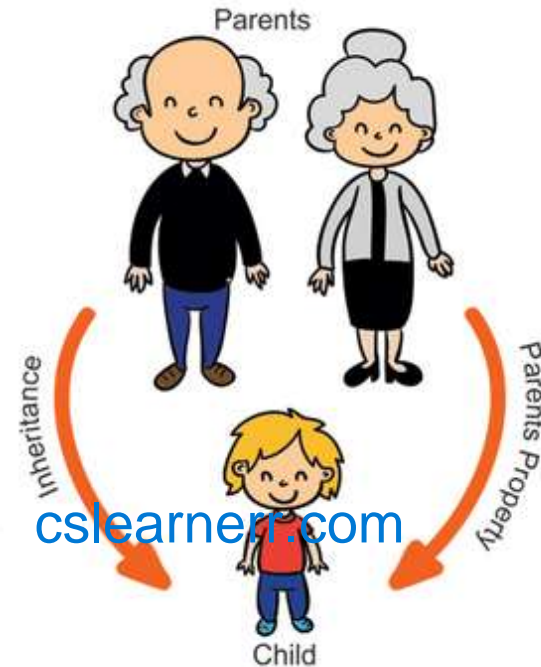
Object Oriented Key Principles

1. Data Abstraction and Encapsulation
2. Inheritance
3. Polymorphism

Discover comprehensive notes for BSCS, BSIT and BSAI courses conveniently gathered on our user-friendly website, www.cslearnerr.com

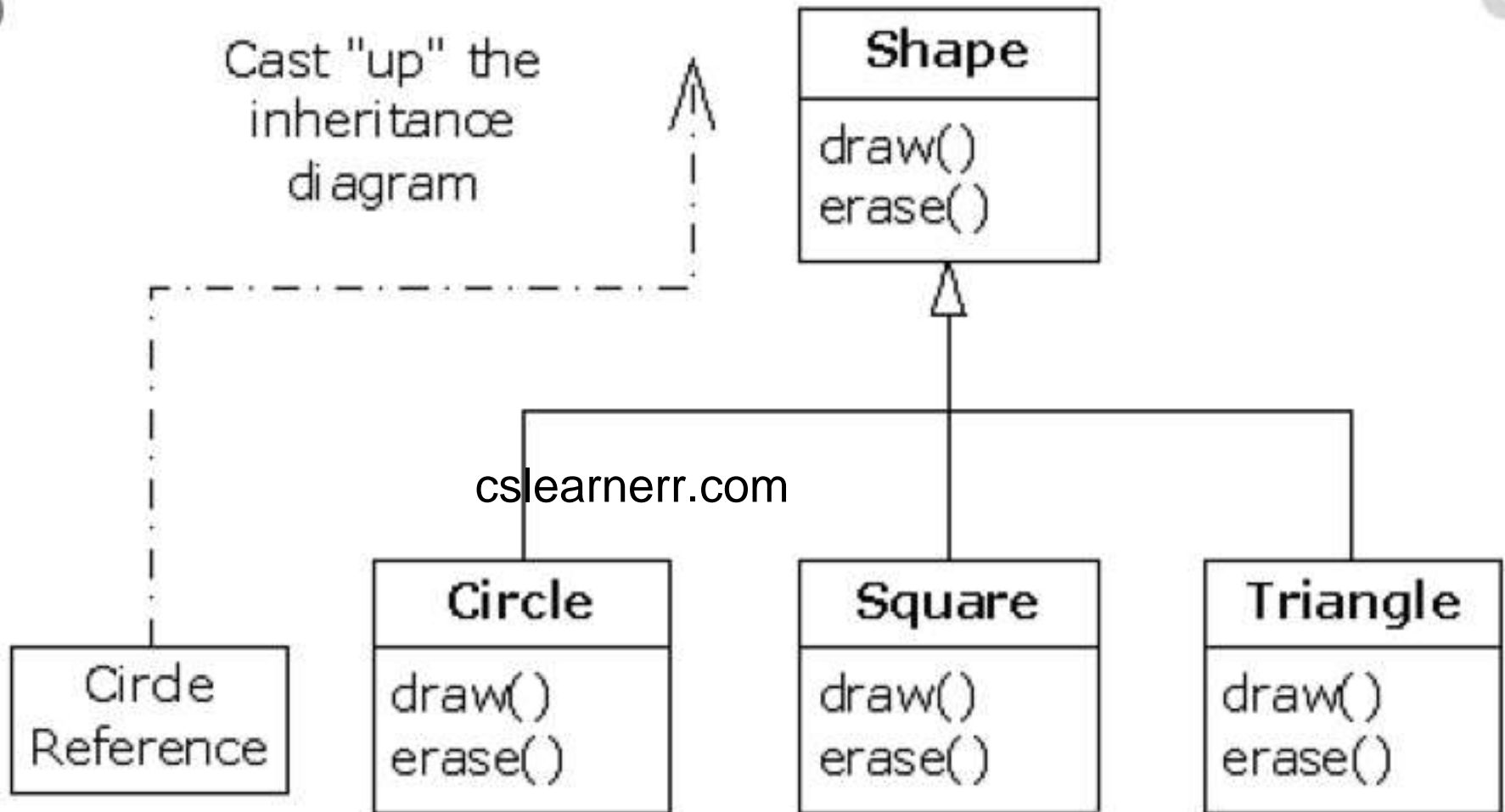
Object Oriented Key Principles

2- Inheritance



Object Oriented Key Principles

2- Inheritance



Inheritance

- A class can extend another class, inheriting all its data members and methods while redefining some of them and/or adding its own.
- Inheritance represents the *is a* relationship between data types. For example: a Circle *is a* Shape.

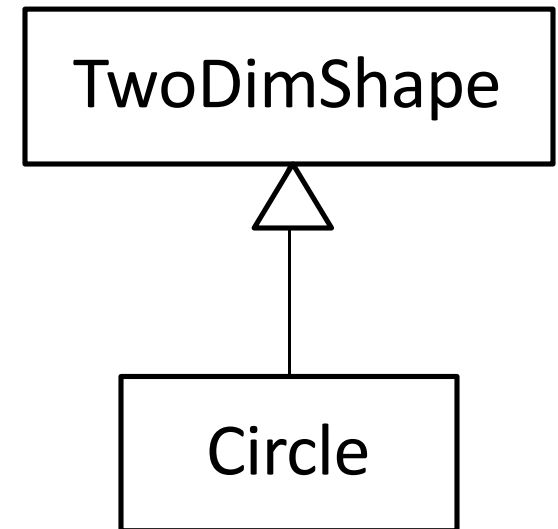
Inheritance in Java:

subclass

extends

superclass

```
public class Circle extends TwoDShape
{
    ...
}
```



cslearnerr.com

Inheritance in Java

- Java supports inheritance
 - A “subclass” inherits from a “superclass”
- In the sub class:
 - In class header use the **extends** keyword to specify the superclass
 - Only declare the additional fields and methods
 - In subclass constructor call the superclass constructor via **super** keyword
 - (**optionally**) override superclass methods

Inheritance Example

```
public class Square {  
    protected int length;  
    public Square(int len){length = len;}  
    public int perimeter(){return 4 * length;}  
    public int area(){return length * length;}  
}
```

```
public class Rectangle extends Square{  
    private int width;  
    public Rectangle(int len, int wid){  
        super(len);  
        width = wid;  
    }  
    public int perimeter()  
        {return 2 * (length + width);}  
    public int area(){return length * width;}  
}
```

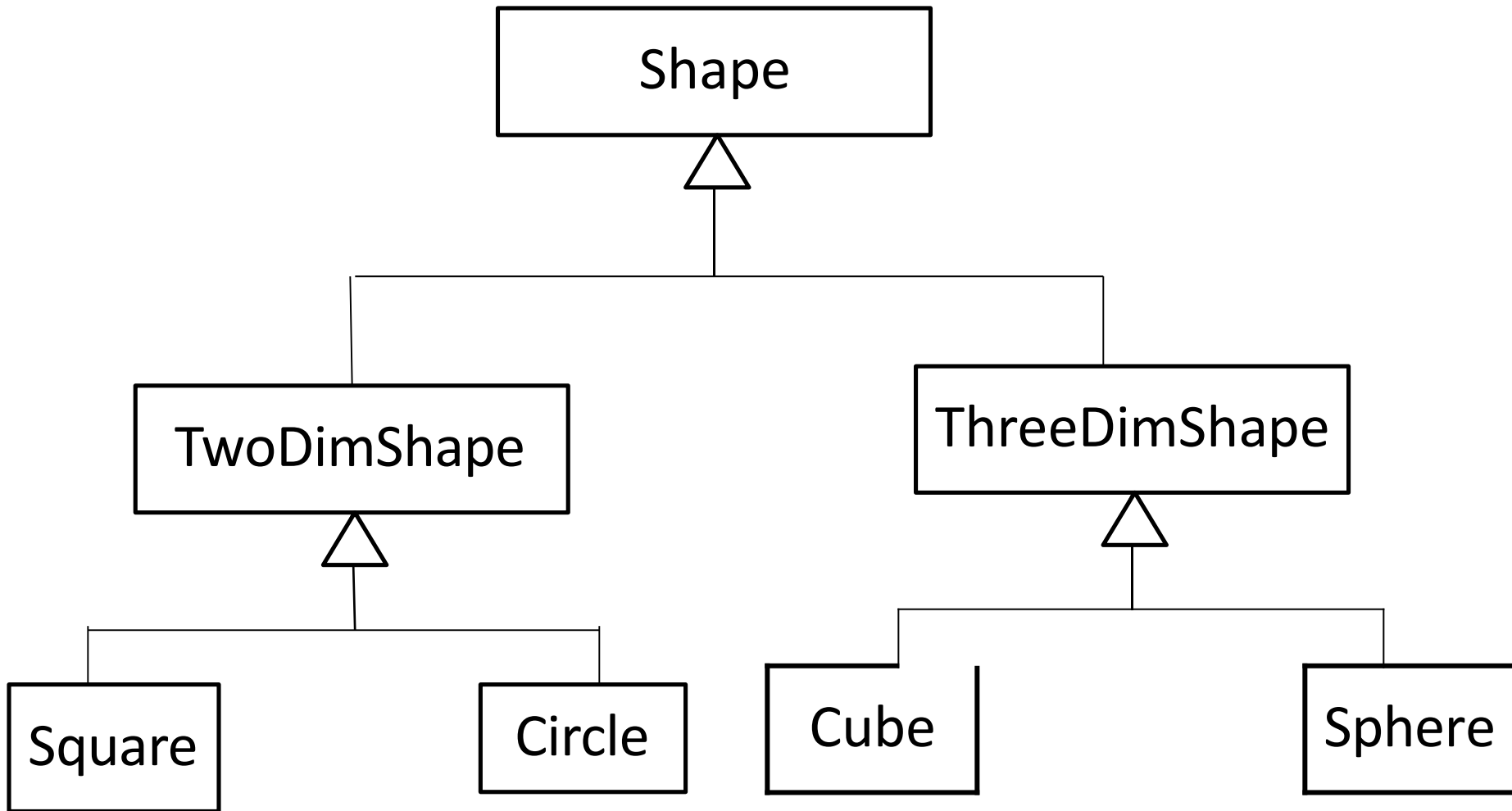

Visibility

- `public`
 - Any field or method specified as `public` can be used by any external class
 - Class constructors, accessors and service methods should be specified as `public`
- `private`
 - Any field or method specified as `private` can be used by code inside the class
 - Fields and internal helper methods should be `private`
- `protected`
 - Any field or method specified as `protected` can be used by code inside the class and in subclasses
 - i.e. classes which inherit from the original class

Multiple Inheritance

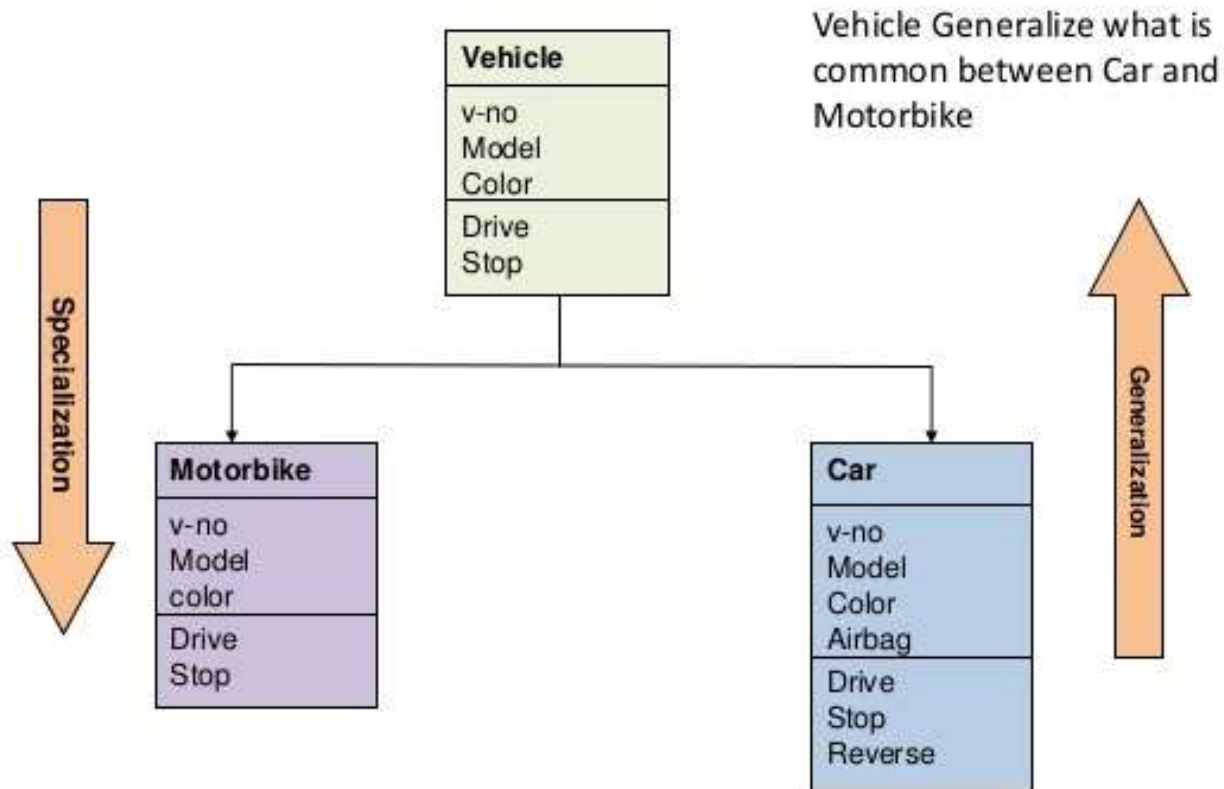
- Multiple inheritance allows a class to be derived from two or more classes, inheriting the members of all **parents**
 - Collisions, such as the same variable name in two parents, have to be resolved
- **Java supports single inheritance**, meaning that a derived class can have only one parent class
- Java does not support multiple inheritance via two or more classes
 - However, multiple ***interfaces inheritance*** is allowed (interfaces in next class)

Inheritance Hierarchy Example



Inheritance Hierarchy Example

Generalization / Specialization

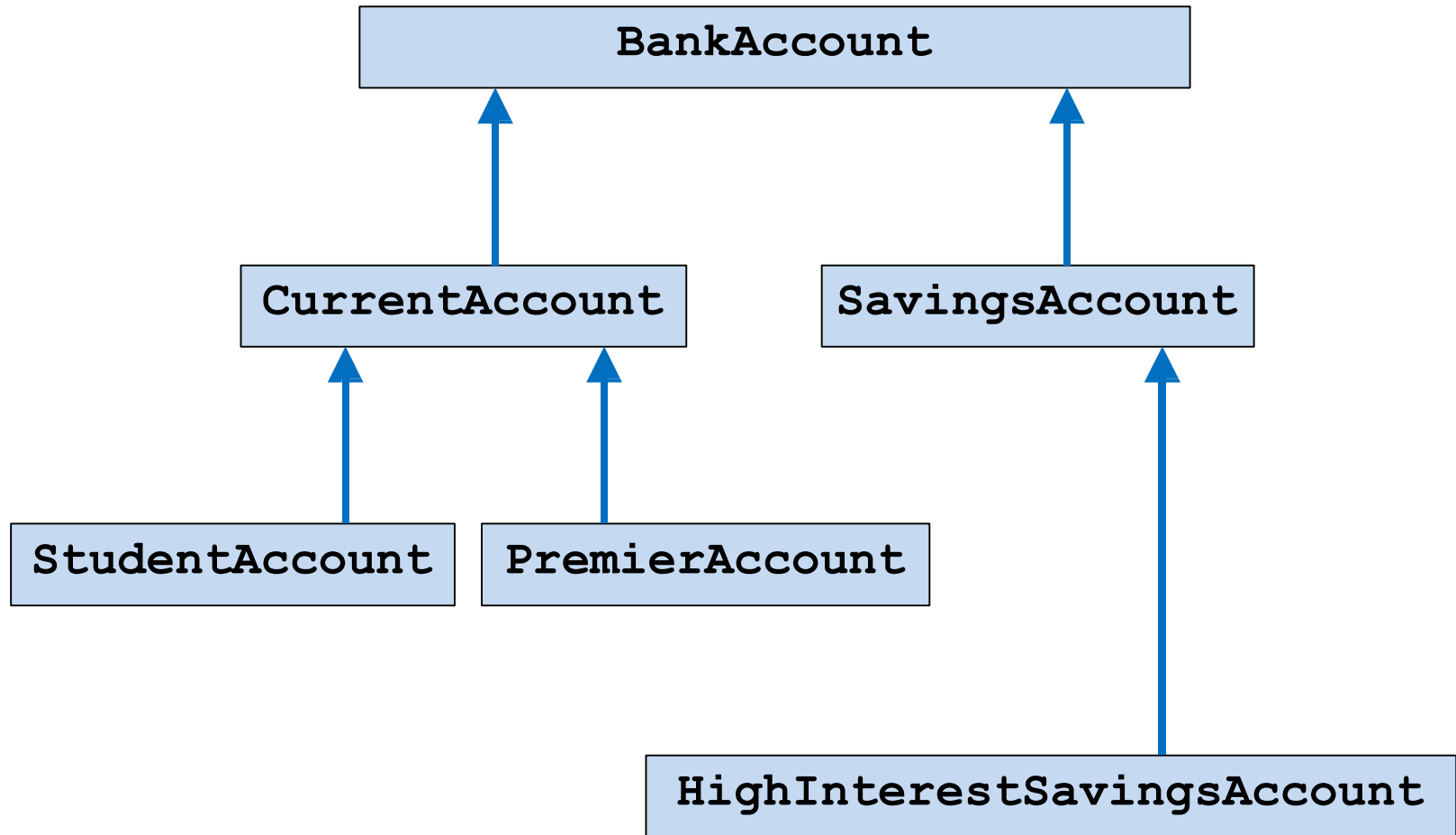


Car and Motorbike specialize Vehicle to their own sub-type.

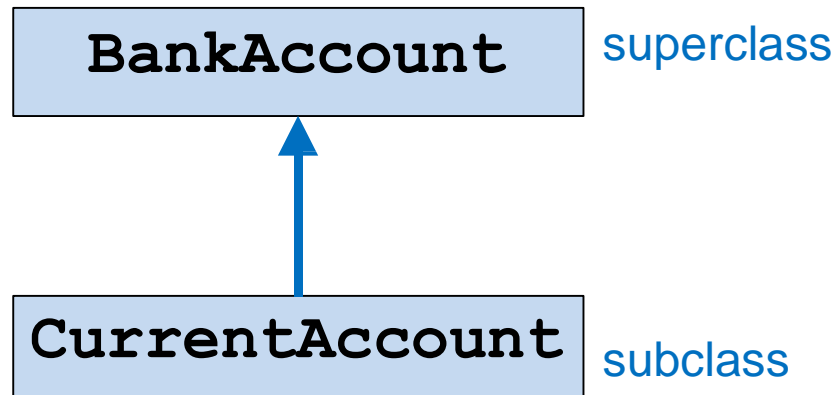
Inheritance Relationships

- Inheritance creates an **is-a** relationship,
 - I.e. the child is a **specialised** version of the parent
- A child class of one parent can be the parent of another child, forming a class hierarchy.
- A subclass will inherit all attributes and operations defined in any of its super classes
 - Subclass may be augmented(to make greater) with additional attributes and operations
 - **Subclass can override attributes and operations**

Inheritance Hierarchy Example



Banking Inheritance Example



- Subclass inherits the variables and methods defined by the super class

```
public class CurrentAccount extends BankAccount {
```

- Subclass specializes by adding its own members:

- fields: `overdraftLimit`
- methods: `getOverdraftLimit()`
`setOverdraftLimit()`
`debit() - overridden`

Overriding methods

- A subclass can override the definition of an inherited method in favour of its own
 - Unless the original method is defined as `final` in the super class
- The new overridden version of the method must have the same signature as the parent's method
 - But can have a different body
- The type of the object executing the method determines which version of the method is invoked
 - The original method in the super class can be invoked explicitly using the `super` reference

Overloading vs. Overriding

- Overloading
 - Multiple methods with the same name in the same class, but with different signatures (**parameters**)
 - Allows similar operation to be defined in different ways for different parameters
- Overriding
 - **Two methods with same name and the same signature**
 - Original version in a parent class
 - Recoded version in a child class
 - Allows a similar operation to be defined in different ways for different sub classes

Superclass – BankAccount (1)

```
public class BankAccount {  
    //fields  
    private String number;  
    private String name;  
    private double balance;  
  
    //default constructor  
    public BankAccount() {  
        number = "-----";  
        name = "-----";  
        balance = 0.0;  
    }  
  
    //overloaded constructor  
    public BankAccount(String accountNo, String accountName) {  
        number = accountNo;  
        name = accountName;  
        balance = 0.0;  
    }  
}
```

Superclass – BankAccount (2)

```
public String getAccountNo() {return number;}
public String getAccountName() {return name;}
public double getBalance() {return balance;}
public void setAccountName(String accountName)
    {name=accountName;}

public void credit (double amount)
    {balance = balance + amount;}
public void debit (double amount)
    {balance = balance - amount;}
}
```

Adding constructors in a subclass

- Super class constructors are not inherited by the sub class, even though they have public visibility
 - However we need to use the super class constructor in order to set up the *"parent's part"* of the object
- The `super` reference is a reference to the super class of a sub class
 - Whereas `this` is a reference to the class itself
- `Super` is used to invoke the parent's constructor

```
super(accountNo, accountName);
```
- The sub class constructor specifies parameters to:
 - Initialise fields from its super class
 - And to initialise its own fields

Subclass Constructors

```
//default constructor
public CurrentAccount(String accountNo,
String accountName) {
    //invoke the parent's constructor
    super(accountNo, accountName);
    //initialise field
    overdraftLimit = 0.0;
}

//overloaded constructor
public CurrentAccount(String accountNo,
String accountName, double accountLimit) {
    super(accountNo, accountName);
    overdraftLimit = accountLimit;
}
```

Inheritance Design Issues (1)

- All derivations should be **is a** relationships
 - i.e. a sub class **is a** child of super class
- Override methods as appropriate to tailor or change the functionality of a child
 - Even if there are no current uses for them, override general methods
- Add new variables to children, but don't redefine (i.e. **do not shadow**) inherited variables
 - Use visibility modifiers carefully to provide needed access without violating encapsulation

Inheritance Design Issues (2)

- Allow each class to manage its own data
 - Use the `super` reference to invoke the parent's constructor to set up its data
- Use the `final` modifier to prevent inheritance
 - If the `final` modifier is applied to a method, then that method cannot be overridden in any descendant classes
 - If the `final` modifier is applied to an entire class, then that class cannot be used to derive any children at all
 - `Final` should be your default choice!

Summary

- A subclass inherits methods and fields from superclass by using the keyword “extends”
- **Multiple Inheritance from classes is not supported in Java**
- Classes form a hierarchy in any application
- A subclass does not inherit constructors
- A subclass can redefine a method specifically for its own needs

Further Reading

- Chapter 9 of Java: How to Program 8e by Dietel & Dietel

Discover comprehensive notes for BSCS, BSIT and BSAI courses conveniently gathered on our user-friendly website, www.cslearnerr.com