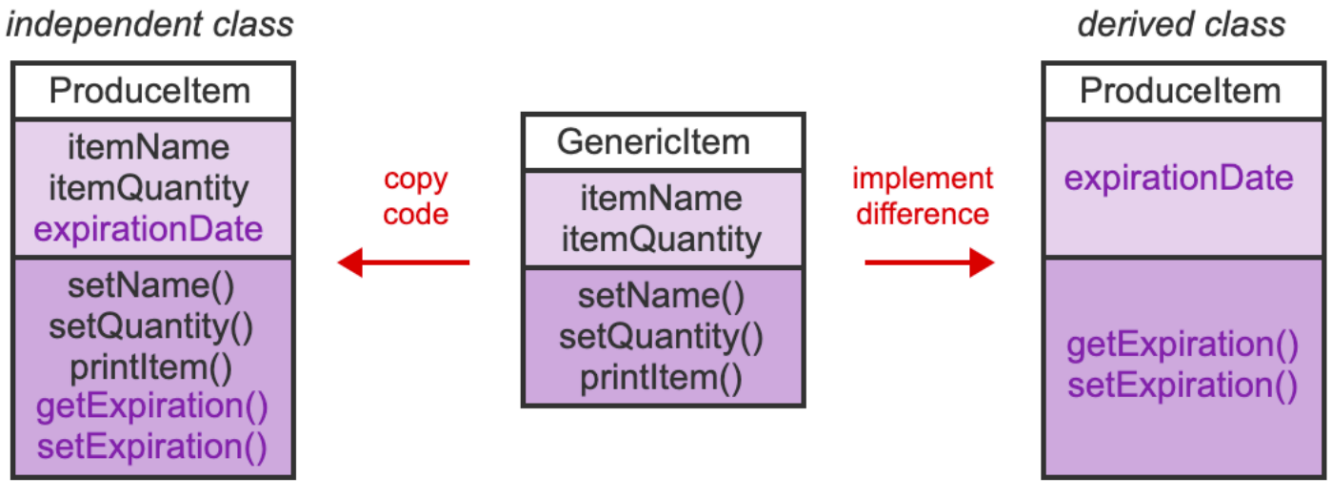
Inheritance

Inheritance is one of the core concepts of object-oriented programming (OOP) languages. It is a mechanism where you can to derive a class from another class for a hierarchy of classes that share a set of attributes and methods.

Derived classes

Concepts

- The main idea is to speed up the code. Reuse the base code, modify, and use it.
- Inheritance works like building blocks.
- One class is similar to another class but with some additions or variations.



Inheritance

- A **derived class** (or **subclass**) is a class that is derived from another class, called a **base class** (or **superclass**).
- The derived class is said to inherit the properties of the base class, a concept called **inheritance**.
- A derived class is declared by placing the keyword **extends** after the derived class name, followed by the base class name.

Example code - base & derived classes

GenericItem.java - base class

```
1  public class GenericItem {
2     private String itemName;
3     private int itemQuantity;
4
5     public void setName(String newName) {
6        itemName = newName;
7     }
8
9     public void setQuantity(int newQty) {
10       itemQuantity = newQty;
11    }
12
13    public void printItem() {
14       System.out.println(itemName + " " + itemQuantity);
15    }
16 }
17
```

ProduceItem.java - derived class

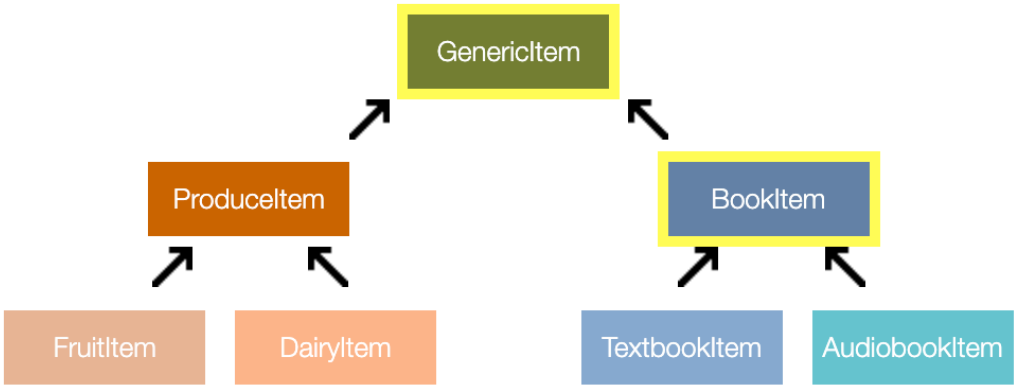
```
public class ProduceItem extends GenericItem {
   private String expirationDate;

   public void setExpiration(String newDate) {
      expirationDate = newDate;
   }

   public String getExpiration() {
      return expirationDate;
   }
}
```

Main ClassDerivationEx.java - main

```
public class ClassDerivationEx {
   public static void main(String[] args) {
      GenericItem miscItem = new GenericItem();
      ProduceItem perishItem = new ProduceItem();

      miscItem.setName("Crunchy Cereal");
      miscItem.setQuantity(9);
      miscItem.printItem();

      perishItem.setName("Apples");
      perishItem.setQuantity(40);
      perishItem.setExpiration("Dec 5, 2019");
      perishItem.printItem();

      System.out.println("  (Expires: "
         + perishItem.getExpiration() + ")");
   }
}
```

Inheritance scenarios

- A derived class can serve as a base class for another class.
- A class can serve as a base class for multiple derived classes.
- A class can only be derived from **one base class directly**_.



`public class TextbookItem extends BookItem`

`public class AudiobookItem extends BookItem`

`public class BookItem extends GenericItem`

`public class GenericItem`

The other left half works similarly, as shown in the example above.

Example code - base & multiple derived classes

Business.java - base class

```java
public class Business {
   private String name;
   private String address;

   public void setName(String busName) {
      name = busName;
   }

   public void setAddress(String busAddress) {
      address = busAddress;
   }

   public String getDescription() {
      return name + " -- " + address;
   }
}
```

Business.java - derived class

```
public class Restaurant extends Business {
   private int rating;

   public void setRating(int userRating) {
      rating = userRating;
   }

   public int getRating() {
      return rating;
   }
}
```

InheritanceExample.java - main

```
public class InheritanceExample {
   public static void main(String[] args) {
      Business someBusiness = new Business();
      Restaurant favoritePlace = new Restaurant();

      someBusiness.setName("ACME");
      someBusiness.setAddress("4 Main St");

      favoritePlace.setName("Friends Cafe");
      favoritePlace.setAddress("500 W 2nd Ave");
      favoritePlace.setRating(5);

      System.out.println(someBusiness.getDescription());
      System.out.println(favoritePlace.getDescription());
      System.out.println("  Rating: " + favoritePlace.getRating());
   }
}
```

Access by members of derived classes

Member access

- Member methods of a derived class cannot access private members of the base class.
- Private is private. It defeats the purpose of a private variable if the private variable is accessible outside of the class.

```java
public class Business {
   private String name;
   private String address;

   ...

}

public class Restaurant extends Business {
   private int rating;

   ...

   public void displayRestaurant() {
      System.out.println(name);
      System.out.println(address);
      System.out.println("Rating: "+ rating);
   }
   ...
}
```

```
$ javac Restaurant.java
Restaurant.java:12: name has private access in Business
      System.out.println(name);
                         ^
Restaurant.java:12: address has private access in Business
      System.out.println(address);
                         ^
2 errors
```

Protected member access

- Provides access to derived classes and all classes in the same **package** but not by anyone else.

Business.java:

```java
public class Business{
   protected String name;    // Member accessible by self and derived classes
   private String address;   // Member accessible only by self

   public void printMembers() { // Member accessible by anyone
      // Print information ...
   }

}
```

Restaurant.java:

```java
public class Restaurant extends Business{
   private int rating;

   public void displayRestaurant() {
      // Attempted accesses
      printMembers();             // OK
      name = "Gyro Hero";         // OK    ("protected" above made this possible)
      address = "5 Fifth St";     // ERROR
   }

   // Other class members ...
}
```

InheritanceAccessEx.java

```
public class InheritanceAccessEx {
   public static void main(String[] args) {
      Business business = new Business();
      Restaurant restaurant = new Restaurant();

      // Attempted accesses
      business.printMembers();          // OK
      business.name = "Gyro Hero";      // OK (protected also applies to other classes in the same package)
      business.address = "5 Fifth St";  // ERROR

      restaurant.printMembers();        // OK
      restaurant.name = "Gyro Hero";    // OK (protected also applies to other classes in the same package)
      restaurant.rating = 5; // ERROR

      // Other instructions ...
   }
}
```

Access specifiers for class members

Specifier	Description
private	Accessible by self.
protected	Accessible by self, derived classes, and other classes in the same package.
public	Accessible by self, derived classes, and everyone else.
no specifier	Accessible by self and other classes in the same package.

Class definitions

Separately, the keyword "public" in a class definition like public class DerivedClass {...} specifies a class's visibility in other classes in the program:

- *public* : A class can be used by every class in the program regardless of the package in which either is defined.
- *no specifier* : A class can be used only in other classes within the same package, known as **package-private**.

Overriding member methods

Overriding

- When a derived class defines a member method that has the same name and parameters as a base class's method, the member method is said to **override** the base class's method.
- The **@Override** annotation is placed above a method that overrides a base class method so the compiler verifies that an identical base class method exists.
- An **annotation** is an optional command beginning with the "@" symbol that can provide the compiler with information that helps the compiler detect errors better.

Example code - overriding member method

Business.java - base class

```java
public class Business {
    protected String name;
    protected String address;

    public void setName(String busName) {
        name = busName;
    }

    public void setAddress(String busAddress) {
        address = busAddress;
    }

    public String getDescription() {         //Same method as in Restaurant class
        return name + " -- " + address;
    }
}
```

Restaurant.java - derived class

```java
public class Restaurant extends Business {
    private int rating;

    public void setRating(int userRating) {
        rating = userRating;
    }

    public int getRating() {
        return rating;
    }

    @Override
    public String getDescription() {            //same method as in Business class
        return name + " -- " + address + " 96422";
    }
}
```

OverridingExample.java - base class

OverridingExample.java - base class

```
public class InheritanceExample {
    public static void main(String[] args) {
        Business someBusiness = new Business();
        Restaurant favoritePlace = new Restaurant();

        someBusiness.setName("ACME");
        someBusiness.setAddress("4 Main St");

        favoritePlace.setName("Friends Cafe");
        favoritePlace.setAddress("500 W 2nd Ave");
        favoritePlace.setRating(5);

        System.out.println(someBusiness.getDescription());
        System.out.println(favoritePlace.getDescription());
        System.out.println("  Rating: " + favoritePlace.getRating());
    }
}
```

> **Overriding vs. Overloading**
>
> Overriding differs from overloading. In overloading, methods with the same name must have different parameter types, number of parameters, or return values. In overriding, a derived class member method must have the same parameter types, number of parameters, and return value as the base class member method with the same name. Overloading is performed if derived and base member methods have different parameter types; the member method of the derived class does not hide the member method of the base class.*

Calling a base class method

An overriding method can call the overridden method by using the super keyword. Ex: `super.getDescription()`. The **super** keyword is a reference variable used to call the parent class's methods or constructors.

Restaurant.java - base class

```
public class Restaurant extends Business {
    private int rating;

    public void setRating(int userRating) {
        rating = userRating;
    }

    public int getRating() {
        return rating;
    }
```

```
    }

    @Override
    public String getDescription() {               //same method as in Business class
        return super.getDescription();
        //return name + " -- " + address + " 96422";  //Zip code added
    }
}
```