# 2021/2022(1)
# IF184301 Object Oriented Programming

Lecture #6

# Inheritance

Misbakhul Munir IRFAN SUBAKTI
司馬伊凡
Мисбакхул Мунир Ирфан Субакти

# Inheritance: Database project example

- Suppose we have three classes below:

DB.java

```
1  package db;
2  abstract class DB {
3      void addSeveral(int keys[]) {
4          for (int i = 0; i < keys.length; i++) {
5              addKey(keys[i]);
6          }
7      }
8      boolean findOneOf(int keys[]) {
9          for (int i = 0; i < keys.length; i++) {
10             if (search(keys[i])) {
11                 return true;
12             }
13         }
14         return false;
15     }
16     abstract void addKey(int key); // No code
17     abstract boolean search(int key); // No code
18 }
```

LinearDB.java

```
1  package db;
2  class LinearDB extends DB {
3      int table[] = new int[10];
4      int size = 0;
5      void addKey(int key) {
6          System.out.println("LinearDB: addKey()");
7      }
8      boolean search(int key) {
9          boolean result = true;
10         System.out.println("LinearDB: searchKey()");
11         return result;
12     }
13 }
```

TreeDB.java

```
1  package db;
2  class TreeDB extends DB {
3      BinarySearchTree data;
4      void addKey(int key) {
5          System.out.println("TreeDB: addKey()");
6      }
7      boolean search(int key) {
8          boolean result = false;
9          System.out.println("TreeDB: searchKey()");
10         return result;
11     }
12 }
```

# Database project example (continued)

- `LinearDB` has 4 operations

```
void addKey(int key)
boolean search(int key)
void addSeveral(int keys[])
boolean findOneOf(int keys[])
```

- It is as if `LinearDB` were defined as:

```
int table[] = new int[10];
int size = 0;
void addKey(int key)
boolean search(int key)
void addSeveral(int keys[])
boolean findOneOf(int keys[])
```

- Similarly for `TreeDB`

# Database project example (continued)

- Variables can be declared of type `DB`. `LinearDB` and `TreeDB` objects can be assigned to them

```
DB ldb = new LinearDB();
DB tdb = new TreeDB();
```

- There are no `DB` objects – we cannot say `new DB()` – only `LinearDB` and `TreeDB` objects

# Database project example (continued)

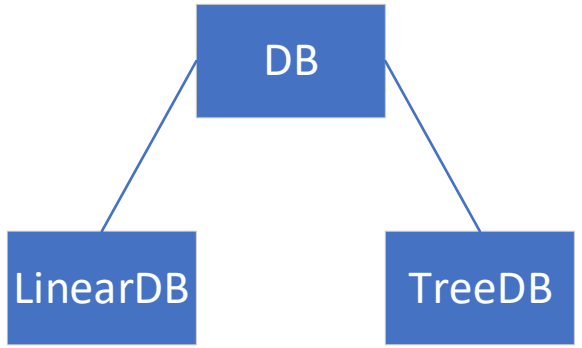
- We can create a test file: `DBTest` as below.

DB.java | LinearDB.java | TreeDB.java | DBTest.java

```
1  package db;
2  public class DBTest {
3      public static void main(String[] args) {
4          DB ldb = new LinearDB();
5          DB tdb = new TreeDB();
6          ldb.addKey(1);
7          if (ldb.search(1)) {
8              System.out.println("It's found!");
9          } else {
10             System.out.println("It's not found!");
11         }
12         tdb.addKey(2);
13         if (tdb.search(1)) {
14             System.out.println("It's found!");
15         } else {
16             System.out.println("It's not found!");
17         }
18     }
19 }
```

# Database project example: Class hierarchy

- `DB` is called the *superclass* or *base class*
- `LinearDB` and `TreeDB` are *subclasses* or *derived classes*
- We say `LinearDB` and `TreeDB` *inherit from* `DB`, because they obtain the definitions of `addSeveral` and `findOneOf`



# Inheritance: General rules from abstract superclasses

- Given

```
abstract class B { … }
abstract C extends B { … }
```

- It is as if `C` were defined as

```
class C { … … }
```

- That is, `C` has all the instance variables and methods defined in `B` in addition to its own instance variables and methods
- If we had `class D extends C { … }`, `D` would *inherit* from both `C` and `B`

# Method redefinition in subclasses

```
abstract class B { … }
abstract C extends B { … }
```

- A very important aspect of inheritance is that *methods can be redefined* instead of being inherited
- Real rule is: `C` *inherits* instance variables of `B` and instance methods of `B`, except those that it defines itself

# Method redefinition

- For example, `LinearDB` could redefine as `addSeveral` as follows.

```
void addSeveral(int keys[]) {
    for (int i = 0; i < keys.length; i++) {
        table[size + i] = keys[i];
    }
    size += keys.length;
}
```

# Inheritance from concrete classes

- It is also possible to inherit from ordinary (i.e., non-abstract, a.k.a. concrete) classes
- Works the same as inheritance from `abstract` classes
- The only difference is that the superclass can have its own instances
- E.g., we could have `TreeDB` inherits from `LinearDB`

# Inheritance from concrete classes (cont'd)

LinearDB.java

```
1  package dbconcrete;
2  class LinearDB {
3      void addKey(int key) {
4          System.out.println("LinearDB: addKey()");
5      }
6      boolean search(int key) {
7          boolean result = true;
8          System.out.println("LinearDB: searchKey()");
9          return result;
10     }
11     void addSeveral(int keys[]) {
12         for (int i = 0; i < keys.length; i++) {
13             addKey(keys[i]);
14         }
15     }
16     boolean findOneOf(int keys[]) {
17         for (int i = 0; i < keys.length; i++) {
18             if (search(keys[i])) {
19                 return true;
20             }
21         }
22         return false;
23     }
24 }
```

LinearDB.java | TreeDB.java

```
1  package dbconcrete;
2  class TreeDB extends LinearDB {
3      BinarySearchTree data;
4      void addKey(int key) {
5          System.out.println("TreeDB: addKey()");
6      }
7      boolean search(int key) {
8          boolean result = false;
9          System.out.println("TreeDB: searchKey()");
10         return result;
11     }
12 }
```

# Inheritance from concrete classes (cont'd)

- But be careful – use of `abstract` classes is usually better!

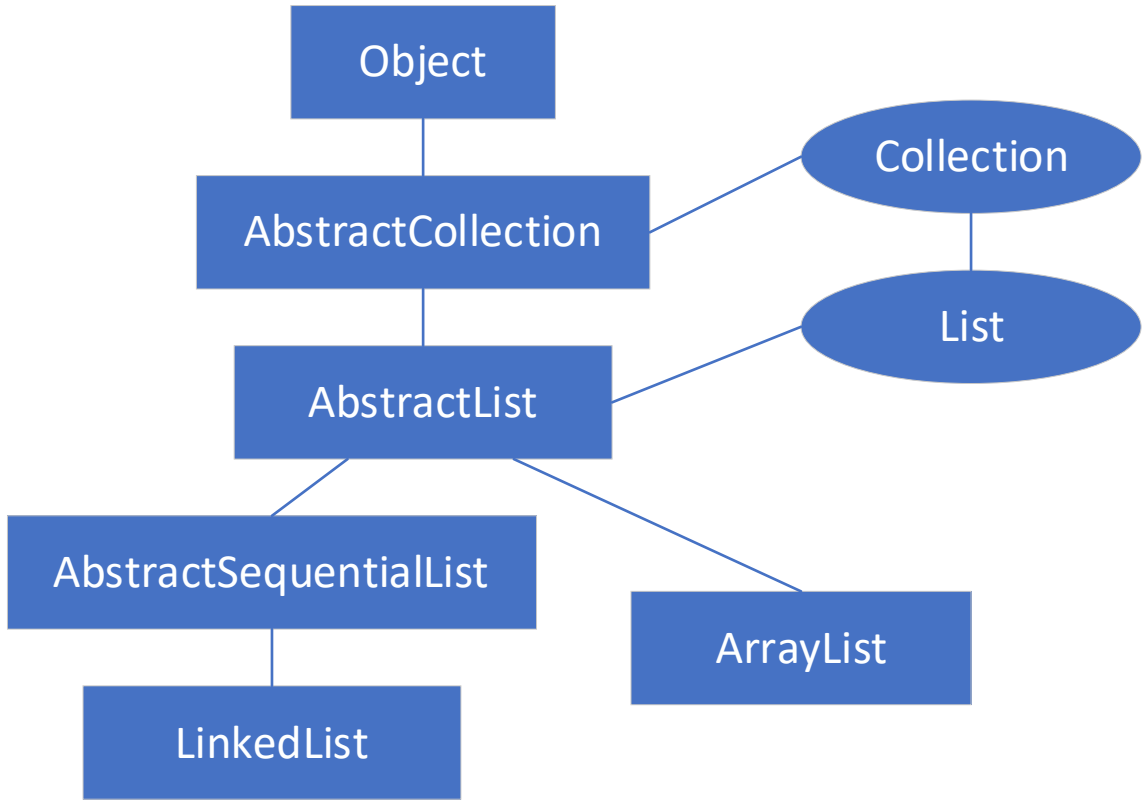
LinearDB.java | TreeDB.java | DBTest.java

```
1  package dbconcrete;
2  public class DBTest {
3      public static void main(String[] args) {
4          LinearDB ldb = new LinearDB();
5          TreeDB tdb = new TreeDB();
6          ldb.addKey(1);
7          if (ldb.search(1)) {
8              System.out.println("It's found!");
9          } else {
10             System.out.println("It's not found!");
11         }
12         tdb.addKey(2);
13         if (tdb.search(1)) {
14             System.out.println("It's found!");
15         } else {
16             System.out.println("It's not found!");
17         }
18     }
19 }
```

# `Interfaces` vs `abstract` classes

- `Interfaces` have only *declarations* for methods
- `Abstract` classes can have both *declarations as well as code* for methods
- But, the constraint is: `abstract` classes only support **single inheritance**
- We **cannot inherit** from **multiple superclasses**
- It is possible to implement **several** `interfaces`

# Inheritance in `Java API`



# Inheritance: Other example

- Geometric figures

```
Figure
   OpenFigure
      Line
      Curve
   ClosedFigure
      Rectangle
      Oval
```

# Inheritance: Other example (continued)

- A class hierarchy for a student information

```
Student
   TaughtStudent
      UndergraduateStudent
      PostgraduateStudent
      OccasionalStudent
   ResearchStudent
```