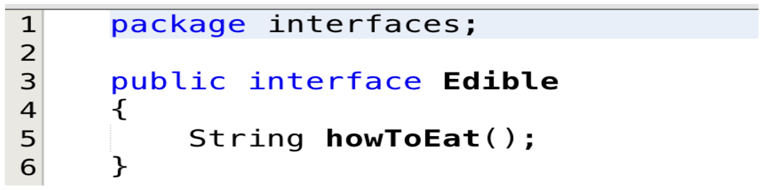
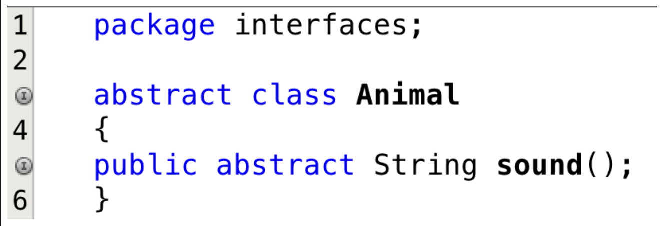
ADSV 2420, Advanced Programming I
Abstract Classes n Interfaces

1. Create project and add a new package called **interfaces**.
2. Under package ***interfaces*** right click and add new Interface Edible

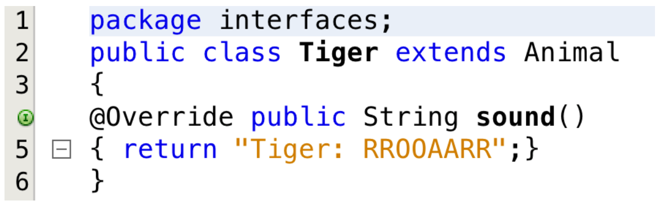


```java
package interfaces;

public interface Edible
{
     String howToEat();
}
```

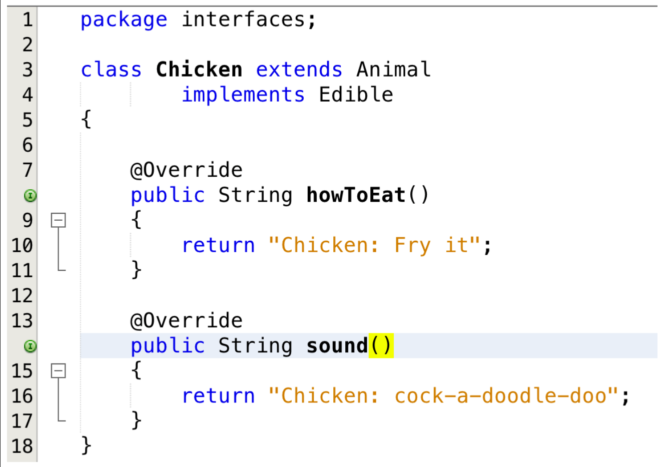
3. Under package ***interfaces*** right click add new classes Animal, Tiger, Chicken, Fruit, Apple and Orange. Observe that class *Fruit* implements interface *Edible* but DOES NOT have to implement the methods of Edible because Fruit is declared abstract. The subclasses of Fruit should implement the methods of Edible or be declared abstract as well.



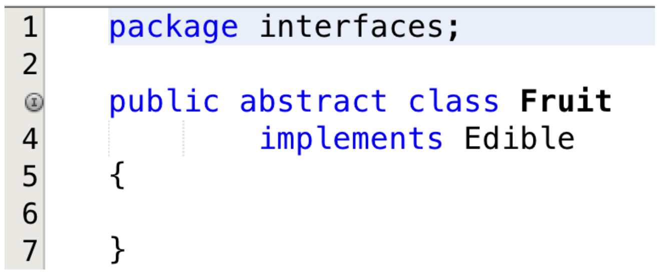
```java
package interfaces;

abstract class Animal
{
public abstract String sound();
}
```



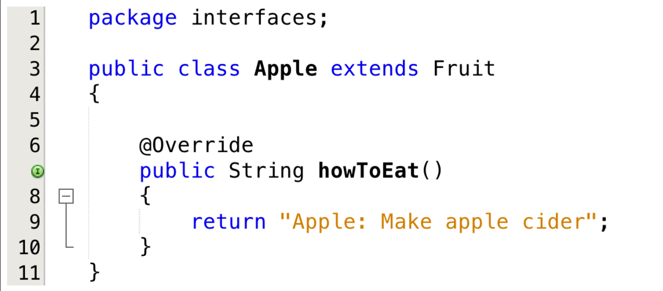
```java
package interfaces;
public class Tiger extends Animal
{
@Override public String sound()
{ return "Tiger: RROOAARR";}
}
```



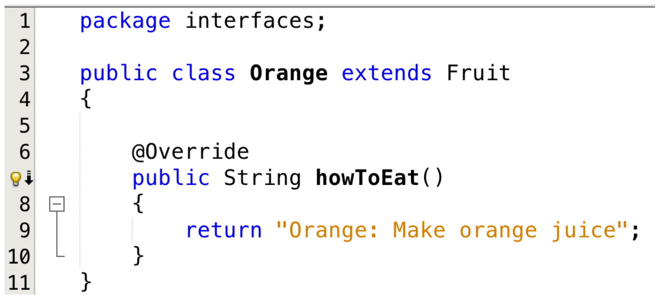
```java
package interfaces;

class Chicken extends Animal
        implements Edible
{

    @Override
    public String howToEat()
    {
        return "Chicken: Fry it";
    }

    @Override
    public String sound()
    {
        return "Chicken: cock-a-doodle-doo";
    }
}
```



```java
package interfaces;

public abstract class Fruit
            implements Edible
{

}
```



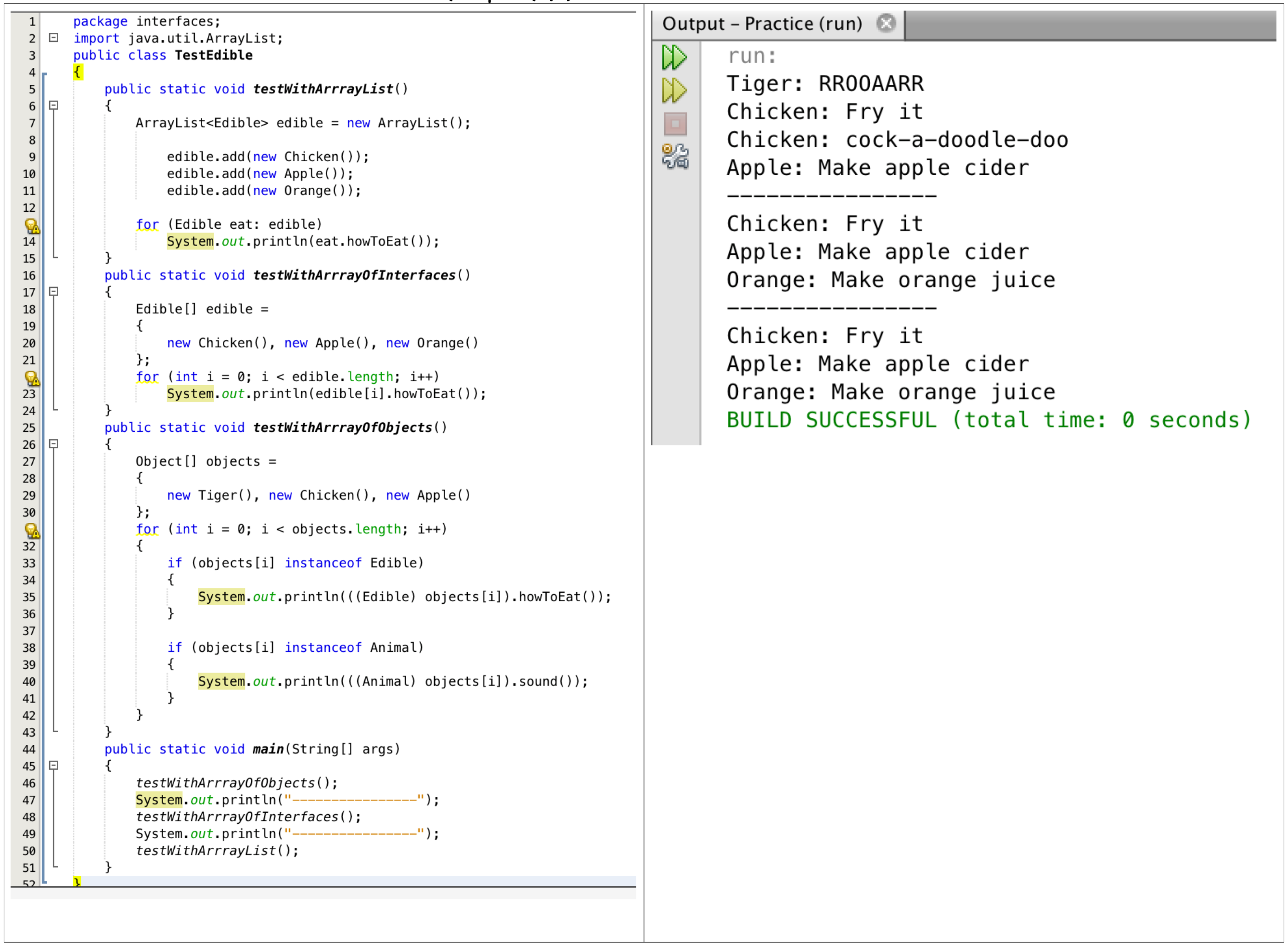
```java
package interfaces;

public class Apple extends Fruit
{

    @Override
    public String howToEat()
    {
        return "Apple: Make apple cider";
    }
}
```



```java
package interfaces;

public class Orange extends Fruit
{

    @Override
    public String howToEat()
    {
        return "Orange: Make orange juice";
    }
}
```

4. Create a class *TestEdible* and test the classes as shown below by
   a) Creating and ArrayList of type *Edible*
   b) Creating an array of type *Edible*
   c) Creating an array of type *Object*

5. *Note the* **instanceof** operator in line 33: Apple is-a Fruit is-a Edible., means the **instanceof** Edible returns true for Apple, Orange, Fruit, Chicken or Tiger. Or in other words, returns true for the subclass and all its ancestors(super(s))



```
 1  package interfaces;
 2  import java.util.ArrayList;
 3  public class TestEdible
 4  {
 5      public static void testWithArrrayList()
 6      {
 7          ArrayList<Edible> edible = new ArrayList();
 8
 9              edible.add(new Chicken());
10              edible.add(new Apple());
11              edible.add(new Orange());
12
13          for (Edible eat: edible)
14              System.out.println(eat.howToEat());
15      }
16      public static void testWithArrrayOfInterfaces()
17      {
18          Edible[] edible =
19          {
20              new Chicken(), new Apple(), new Orange()
21          };
22          for (int i = 0; i < edible.length; i++)
23              System.out.println(edible[i].howToEat());
24      }
25      public static void testWithArrrayOfObjects()
26      {
27          Object[] objects =
28          {
29              new Tiger(), new Chicken(), new Apple()
30          };
31          for (int i = 0; i < objects.length; i++)
32          {
33              if (objects[i] instanceof Edible)
34              {
35                  System.out.println(((Edible) objects[i]).howToEat());
36              }
37
38              if (objects[i] instanceof Animal)
39              {
40                  System.out.println(((Animal) objects[i]).sound());
41              }
42          }
43      }
44      public static void main(String[] args)
45      {
46          testWithArrrayOfObjects();
47          System.out.println("----------------");
48          testWithArrrayOfInterfaces();
49          System.out.println("----------------");
50          testWithArrrayList();
51      }
52  }
```

Output – Practice (run)

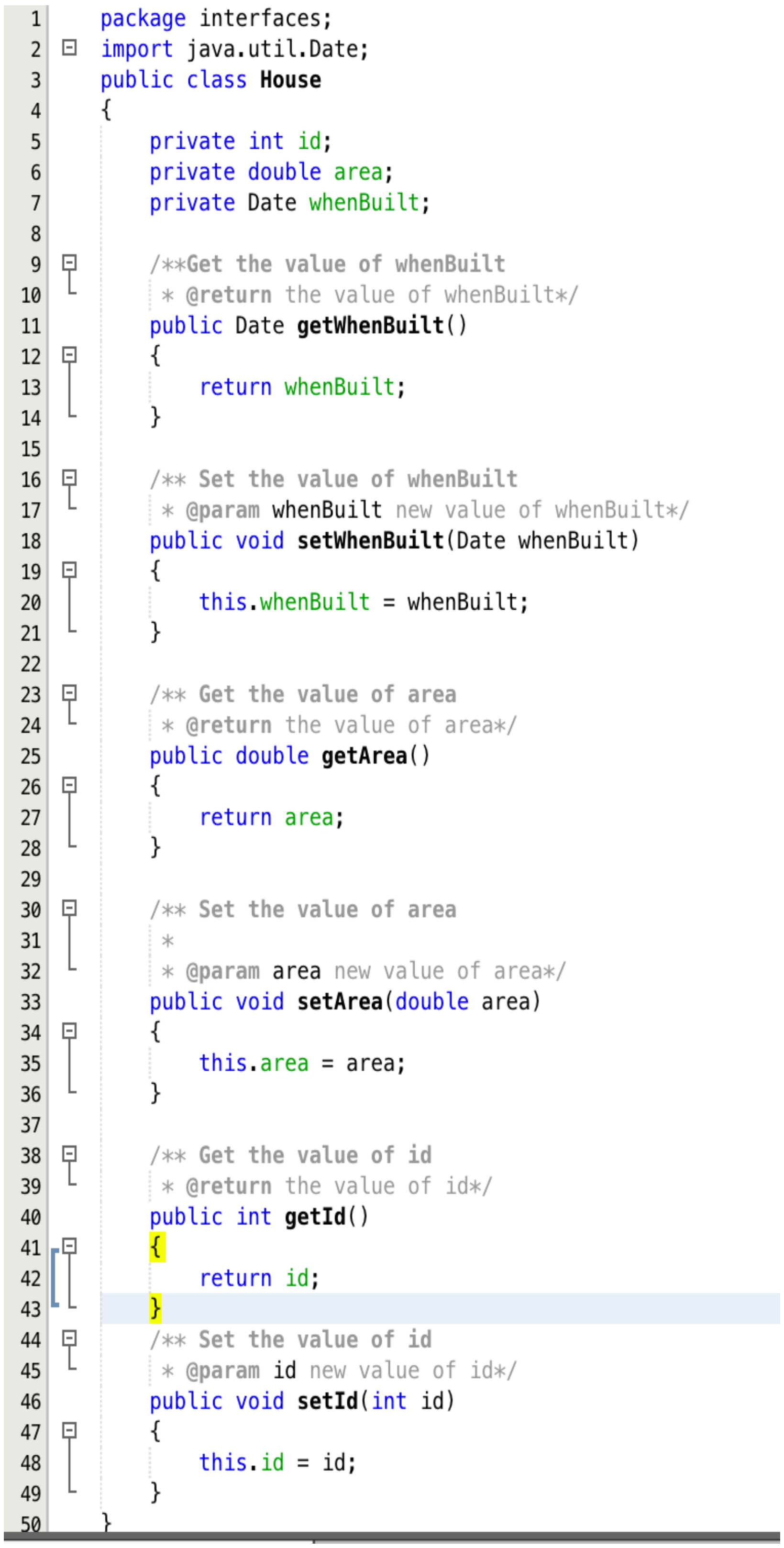
```
run:
Tiger: RROOAARR
Chicken: Fry it
Chicken: cock-a-doodle-doo
Apple: Make apple cider
----------------
Chicken: Fry it
Apple: Make apple cider
Orange: Make orange juice
----------------
Chicken: Fry it
Apple: Make apple cider
Orange: Make orange juice
BUILD SUCCESSFUL (total time: 0 seconds)
```

7. Add class **Duck** that extends Animal and implements Edible. Its method howToEat returns "Duck: Roast it" and its method sound retuns "Duck: quack! quack! Quak!".

8. Add class **Broccoli** that extends Fruit and implements Edible indirectly. Its method howToEat returns "Brocolli: Steam it" .

9. Create new Objects inside the three methods *testWithArrrayList*(), *testWithArrrayOfInterfaces*() and *testWithArrrayOfObjects*() and test your code.

10. Create a new class called *House*. Use Netbeans Insert code *add properties* to create 3 instance variables, **id**, **area** and **whenBuilt** with setters and getters and javadoc as shown below.

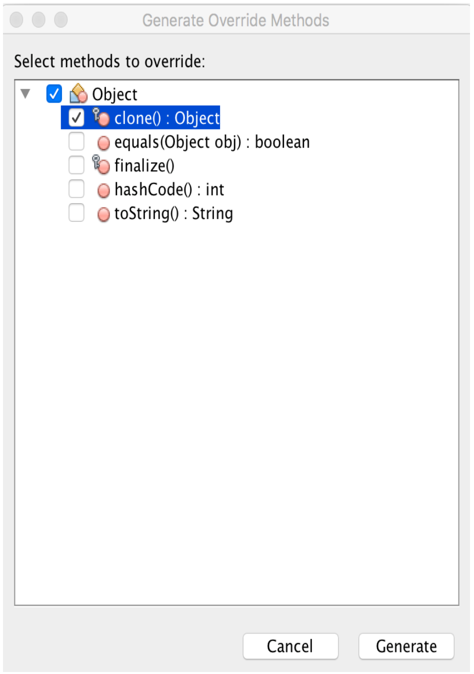


```java
package interfaces;
import java.util.Date;
public class House
{
    private int id;
    private double area;
    private Date whenBuilt;

    /**Get the value of whenBuilt
     * @return the value of whenBuilt*/
    public Date getWhenBuilt()
    {
        return whenBuilt;
    }

    /** Set the value of whenBuilt
     * @param whenBuilt new value of whenBuilt*/
    public void setWhenBuilt(Date whenBuilt)
    {
        this.whenBuilt = whenBuilt;
    }

    /** Get the value of area
     * @return the value of area*/
    public double getArea()
    {
        return area;
    }

    /** Set the value of area
     *
     * @param area new value of area*/
    public void setArea(double area)
    {
        this.area = area;
    }

    /** Get the value of id
     * @return the value of id*/
    public int getId()
    {
        return id;
    }
    /** Set the value of id
     * @param id new value of id*/
    public void setId(int id)
    {
        this.id = id;
    }
}
```

11. Use Netbeans Insert Code and add the constructor shown below. Add the date of building the house, manually as show in line 13.

```
3     public class House
4     {
5         private int id;
6         private double area;
7         private Date whenBuilt;
8 
9         public House(int id, double area)
10        {
11            this.id = id;
12            this.area = area;
13            this.whenBuilt = new Date();
14        }
```

12. Use Netbeans Insert Code, *Override Method* and override the clone of class Object.


Generate Override Methods

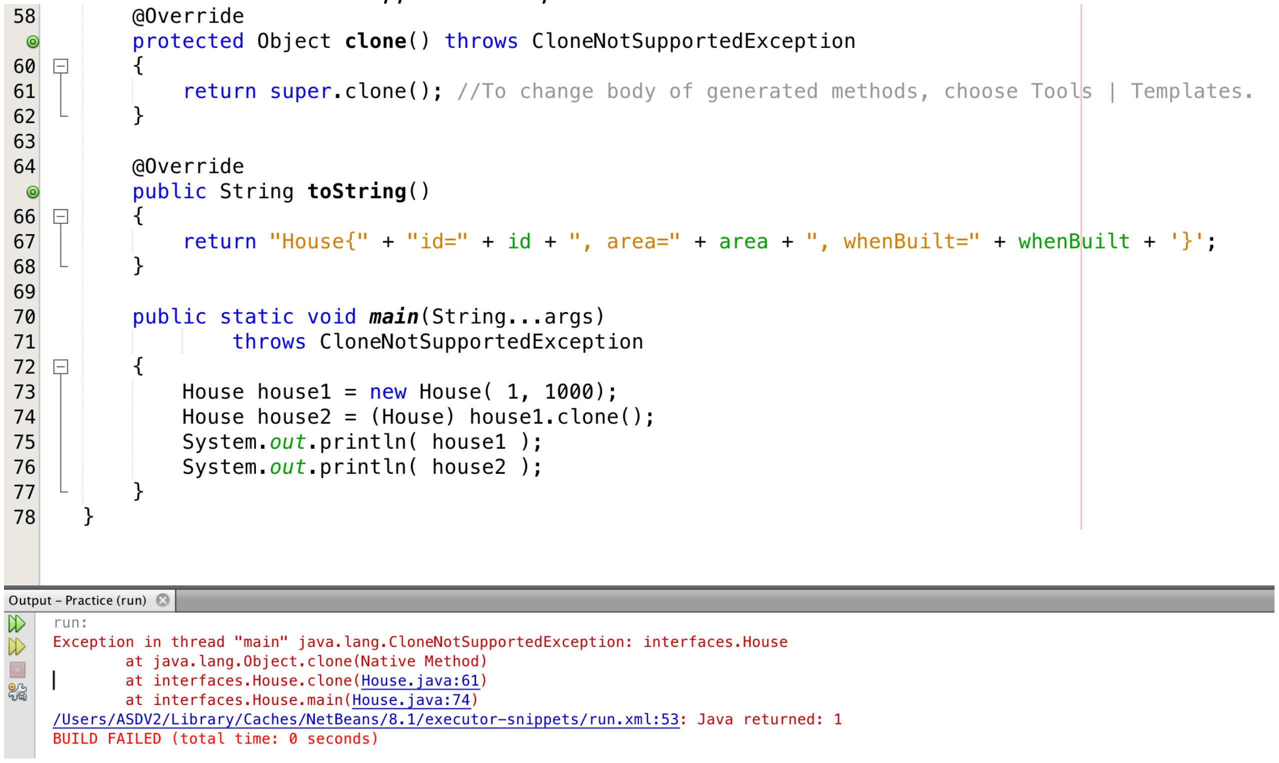
Select methods to override:

- [✓] Object
  - [✓] clone() : Object
  - [ ] equals(Object obj) : boolean
  - [ ] finalize()
  - [ ] hashCode() : int
  - [ ] toString() : String

13. Your code should look as a shown below:

```
    @Override
    protected Object clone() throws CloneNotSupportedException
    {
        return super.clone(); //To change body of generated methods,
    }
```

14. Use Netbeans Insert Code, toString and and the toString as shown below. Then in your main create one house *house1* and clone it into *house2*. Print them inside main(). The program crashes and throws a *CloneNotSupportedException.*

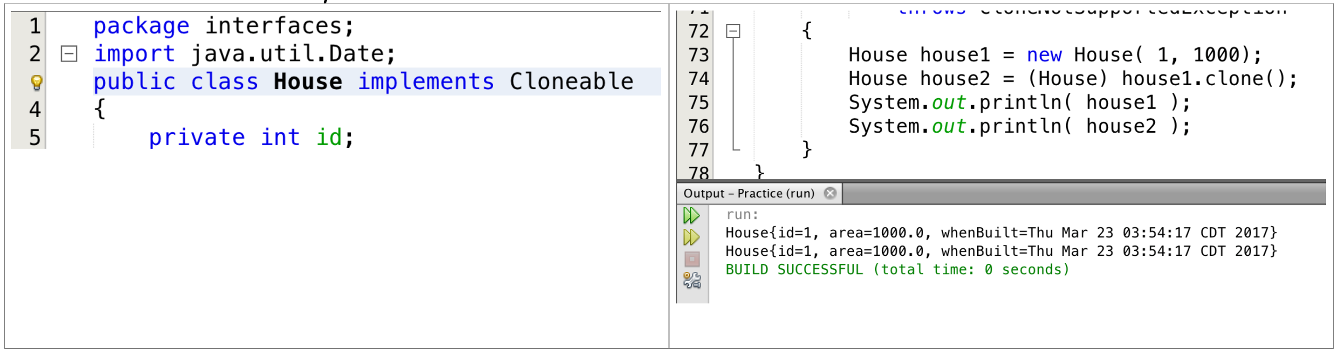


```
58        @Override
          protected Object clone() throws CloneNotSupportedException
60        {
61            return super.clone(); //To change body of generated methods, choose Tools | Templates.
62        }
63
64        @Override
          public String toString()
66        {
67            return "House{" + "id=" + id + ", area=" + area + ", whenBuilt=" + whenBuilt + '}';
68        }
69
70        public static void main(String...args)
71                throws CloneNotSupportedException
72        {
73            House house1 = new House( 1, 1000);
74            House house2 = (House) house1.clone();
75            System.out.println( house1 );
76            System.out.println( house2 );
77        }
78    }
```

Output - Practice (run)

```
run:
Exception in thread "main" java.lang.CloneNotSupportedException: interfaces.House
        at java.lang.Object.clone(Native Method)
        at interfaces.House.clone(House.java:61)
        at interfaces.House.main(House.java:74)
/Users/ASDV2/Library/Caches/NetBeans/8.1/executor-snippets/run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

15. To make the super class of class House (Object) to generate a cloned House add *implements* **Cloneable** and run it again to obtain the output shown below. The OBJECT class gives you a SHALLOW COPY of the house. That is, the *Date whenBuilt **is NOT duplicated,** only primitives area and id are duplicated.*



```
1   package interfaces;
2   import java.util.Date;
    public class House implements Cloneable
4   {
5       private int id;
```

```
71              throws CloneNotSupportedException
72      {
73          House house1 = new House( 1, 1000);
74          House house2 = (House) house1.clone();
75          System.out.println( house1 );
76          System.out.println( house2 );
77      }
78  }
```

Output - Practice (run)

```
run:
House{id=1, area=1000.0, whenBuilt=Thu Mar 23 03:54:17 CDT 2017}
House{id=1, area=1000.0, whenBuilt=Thu Mar 23 03:54:17 CDT 2017}
BUILD SUCCESSFUL (total time: 0 seconds)
```

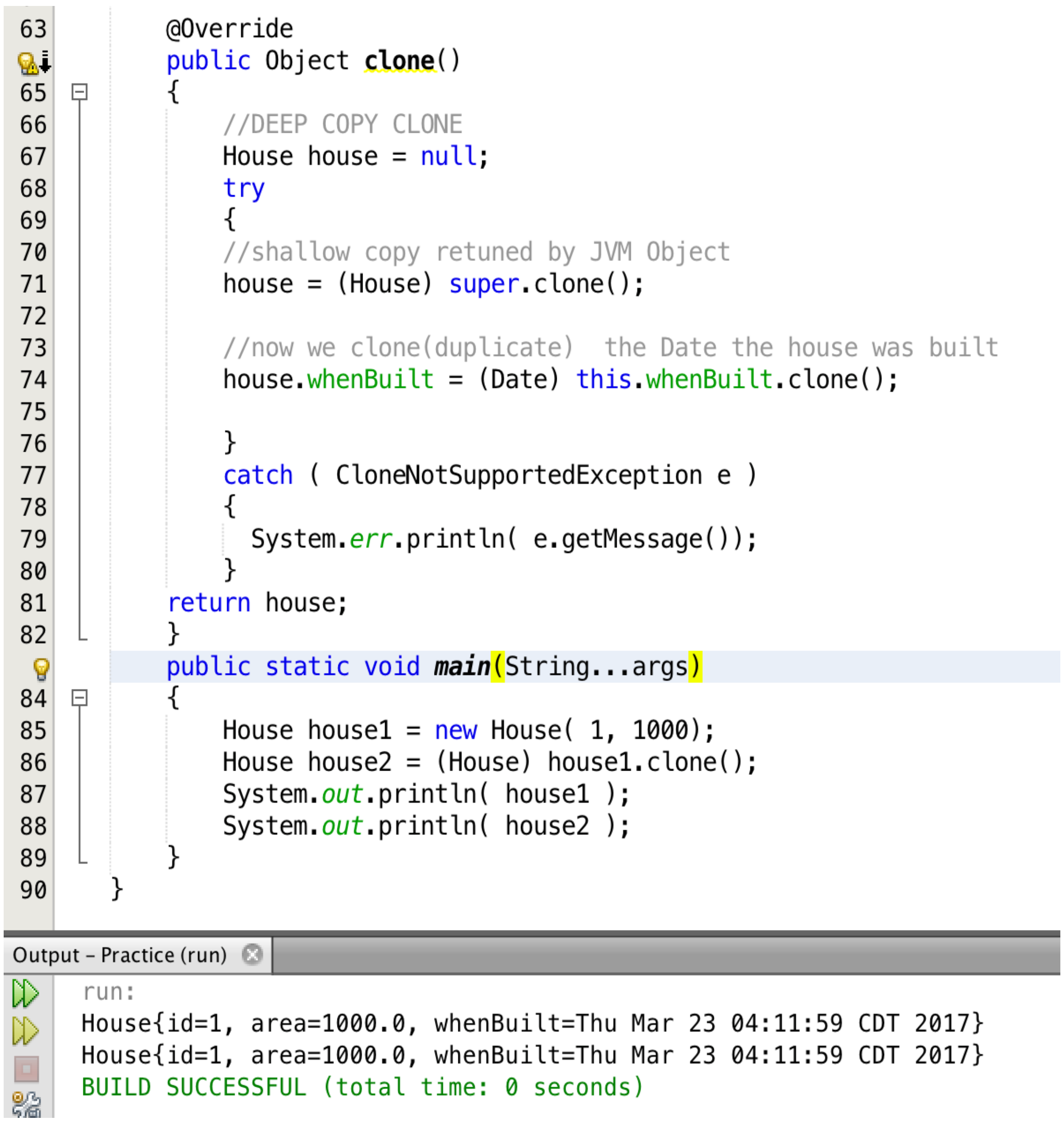
16. We will modify clone method to give us DEEP COPY cloning (duplicate the Date).

a) Make the clone method **public. Yes, we can widen from protected to public, not narrow.**

b) Duplicate the Date the house was built as shown in line 77.

c) No need to declare we that we throw and *CloneNotSupportedException* in either clone() or main() since we handle the exception locally inside method clone().

d) Run the program

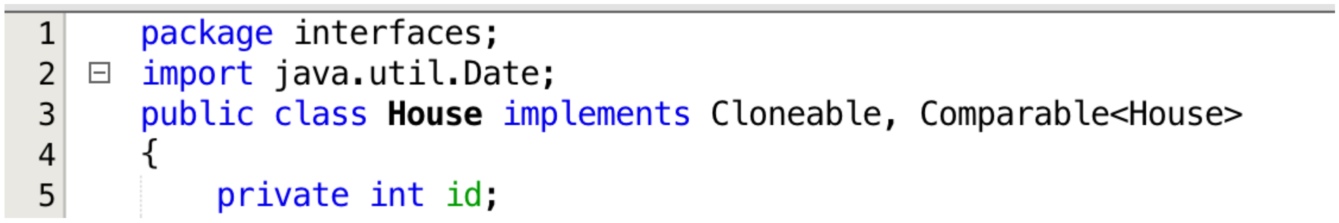


```
63        @Override
          public Object clone()
65        {
66            //DEEP COPY CLONE
67            House house = null;
68            try
69            {
70            //shallow copy retuned by JVM Object
71            house = (House) super.clone();
72
73            //now we clone(duplicate)  the Date the house was built
74            house.whenBuilt = (Date) this.whenBuilt.clone();
75
76            }
77            catch ( CloneNotSupportedException e )
78            {
79              System.err.println( e.getMessage());
80            }
81        return house;
82        }
          public static void main(String...args)
84        {
85            House house1 = new House( 1, 1000);
86            House house2 = (House) house1.clone();
87            System.out.println( house1 );
88            System.out.println( house2 );
89        }
90    }
```

Output – Practice (run)

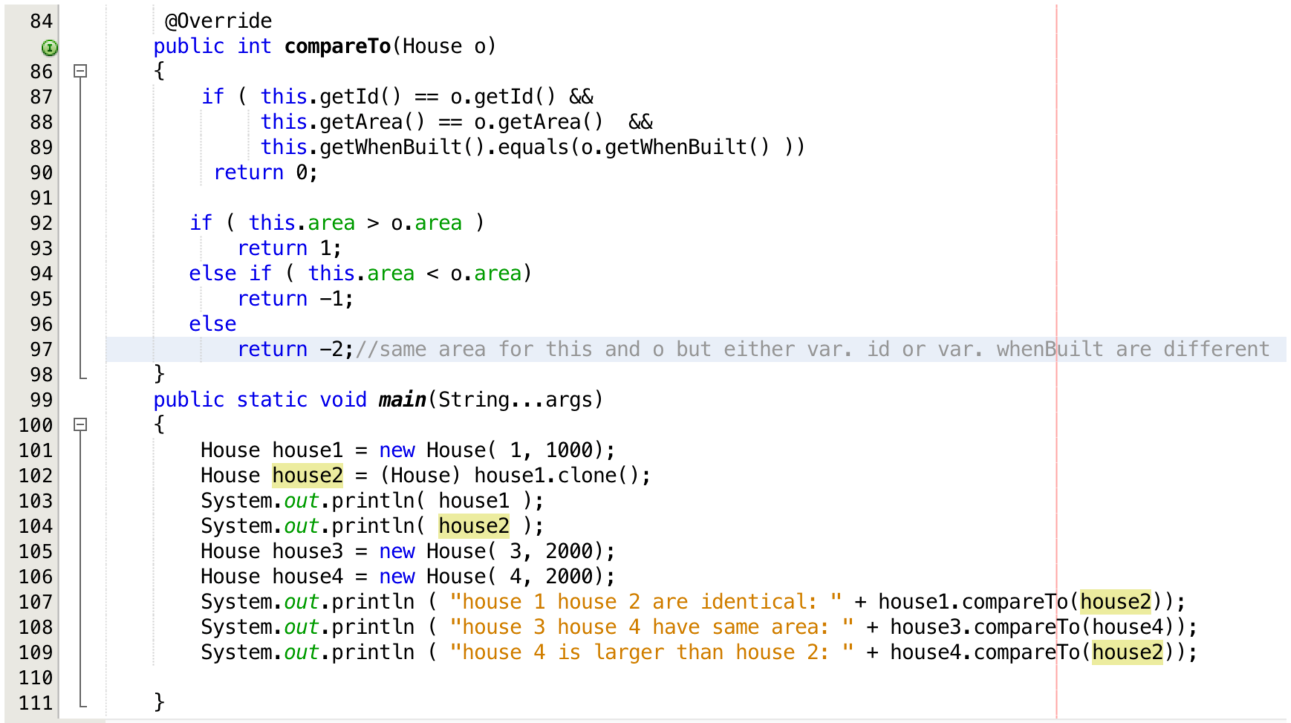
```
run:
House{id=1, area=1000.0, whenBuilt=Thu Mar 23 04:11:59 CDT 2017}
House{id=1, area=1000.0, whenBuilt=Thu Mar 23 04:11:59 CDT 2017}
BUILD SUCCESSFUL (total time: 0 seconds)
```

17. Add Interface *Comparable* to the class as shown, and hit the bulb to *implement all abstract methods.*

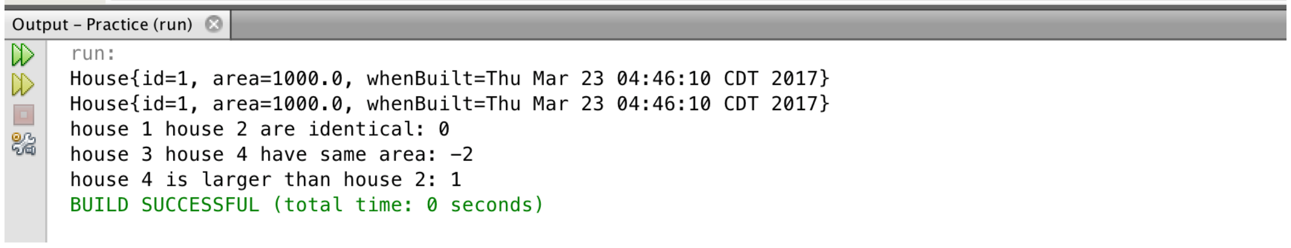


```
package interfaces;
import java.util.Date;
public class House implements Cloneable, Comparable<House>
{
    private int id;
```

18. Implement the *compareTo* as shown. Test it as shown by adding lines 105 to 109 in your main.
1. return 0 when 2 houses are identical,
2. return 1 if the area of *this* is greater than the area of the house of the parameter
3. return -1 if the area of *this* is smaller than the area of the house of the parameter
4. return -2 if the area of *this* is equal to than the area of the house of the parameter but either the Ids are different or the dates the houses were built are different.



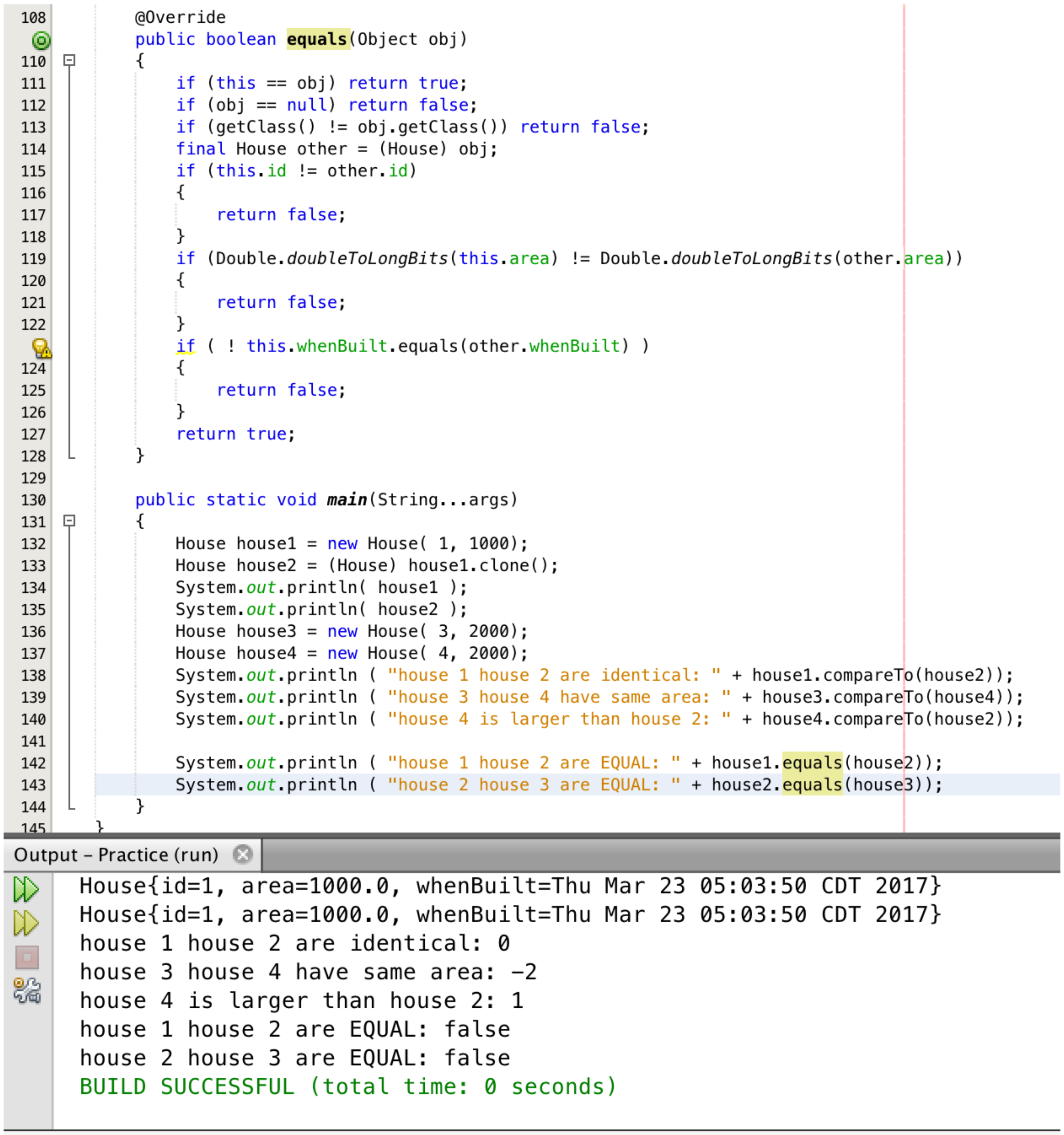
```
 84          @Override
            public int compareTo(House o)
 86         {
 87             if ( this.getId() == o.getId() &&
 88                  this.getArea() == o.getArea()   &&
 89                  this.getWhenBuilt().equals(o.getWhenBuilt() ))
 90              return 0;
 91 
 92            if ( this.area > o.area )
 93                return 1;
 94            else if ( this.area < o.area)
 95                return -1;
 96            else
 97                return -2;//same area for this and o but either var. id or var. whenBuilt are different
 98         }
 99         public static void main(String...args)
100         {
101             House house1 = new House( 1, 1000);
102             House house2 = (House) house1.clone();
103             System.out.println( house1 );
104             System.out.println( house2 );
105             House house3 = new House( 3, 2000);
106             House house4 = new House( 4, 2000);
107             System.out.println ( "house 1 house 2 are identical: " + house1.compareTo(house2));
108             System.out.println ( "house 3 house 4 have same area: " + house3.compareTo(house4));
109             System.out.println ( "house 4 is larger than house 2: " + house4.compareTo(house2));
110 
111         }
```



Output - Practice (run)

```
run:
House{id=1, area=1000.0, whenBuilt=Thu Mar 23 04:46:10 CDT 2017}
House{id=1, area=1000.0, whenBuilt=Thu Mar 23 04:46:10 CDT 2017}
house 1 house 2 are identical: 0
house 3 house 4 have same area: -2
house 4 is larger than house 2: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

19. Use Netbeans Insert Code *Override Method >equals and hashcode* and add the equals method. In line 123, modify the code provided by Netbeans. Test the equals method by adding in main() lines 141, 142.



```
108      @Override
         public boolean equals(Object obj)
110      {
111          if (this == obj) return true;
112          if (obj == null) return false;
113          if (getClass() != obj.getClass()) return false;
114          final House other = (House) obj;
115          if (this.id != other.id)
116          {
117              return false;
118          }
119          if (Double.doubleToLongBits(this.area) != Double.doubleToLongBits(other.area))
120          {
121              return false;
122          }
             if ( ! this.whenBuilt.equals(other.whenBuilt) )
124          {
125              return false;
126          }
127          return true;
128      }
129
130      public static void main(String...args)
131      {
132          House house1 = new House( 1, 1000);
133          House house2 = (House) house1.clone();
134          System.out.println( house1 );
135          System.out.println( house2 );
136          House house3 = new House( 3, 2000);
137          House house4 = new House( 4, 2000);
138          System.out.println ( "house 1 house 2 are identical: " + house1.compareTo(house2));
139          System.out.println ( "house 3 house 4 have same area: " + house3.compareTo(house4));
140          System.out.println ( "house 4 is larger than house 2: " + house4.compareTo(house2));
141
142          System.out.println ( "house 1 house 2 are EQUAL: " + house1.equals(house2));
143          System.out.println ( "house 2 house 3 are EQUAL: " + house2.equals(house3));
144      }
145  }
```

```
House{id=1, area=1000.0, whenBuilt=Thu Mar 23 05:03:50 CDT 2017}
House{id=1, area=1000.0, whenBuilt=Thu Mar 23 05:03:50 CDT 2017}
house 1 house 2 are identical: 0
house 3 house 4 have same area: -2
house 4 is larger than house 2: 1
house 1 house 2 are EQUAL: false
house 2 house 3 are EQUAL: false
BUILD SUCCESSFUL (total time: 0 seconds)
```