CPE203

Java Class

Hadi Asemi
Jul 18,2020

Contents

Definitions:

- **OOP** is a programing technique that focuses on the data(=objects) and on the interfaces to that object.
- **Object:** Is the instance of the class
- **Object Class:** is the ***ultimate super class***. Every class you ever make in Jave will automatically inherit from it.(equals, toString, and getClass)
- **Classes:** A class is the template or blueprint from which objects are made.
  - Static variable:
    - Belong to the class
    - Accessed by: ClassName.VARIABLE_NAME
- **Superclass:** class above another in a hierarchy of classses
- **Subclass:** inherits form parent and is a version of the parent class
- **Encapsulation(sometimes called information hiding):** is simply combining data and behavior in one package hiding the implementation details from the users of the objects.
- **Instance variables:** are variables defined in a class, but outside the body of methods. Instance variables are filled when each object is instantiated and belong the object.
- **Class Variables:** belongs to the class and the value in that variable is shared by every instance of the class by the class itself.
- **Constructor:** set data's values
- **Methods:** Manipulate and access data
- **Public:** is keyword which declares a member's access as public.
- **Private:** is a Java keyword which declares a member's access as private.
- **Static:** object belongs specifically to the class, instead of instances of that class.
- **Is-a:** inheritance/interfaces
- **has-a:** composition/aggresgation

Three key characteristics of objects:

- **The object's behavior**
- **The object's state**
- **The object's identity**

Array:

```java
int[]array= new int[20]; // allocating memory for array. it will be fixed sized
array[0]=5; //declaring first element array

//for loop - if nums is Array
for (int i = 0; i < nums.length i++){
  //nums[i];
}

//for each loop
for (int i: nums){
  //do stuff with i
}
```

ArrayList:

```
ArrayList<String>words=new ArrayList<String>();

ArrayList<Integer>num=new ArrayList<Integer>();

LinkedList<String> ll = new LinkedList<>();

num.add(1);

num.get(0); // we need put index

//remove element base on index
num.remove(1);


//for loop - if nums is ArrayList
for (int i = 0; i < nums.size(); i++){
  //nums.get(i);
}

//for each loop
for (int i: nums){
  //do stuff with i
}
```

HashMap:

```
Map<String,String>myMap=new HashMap<>();

// add key
myMap.put("Hadi","21");

// get the value of the key
myMap.get("Hadi");

// remove the key
myMap.remove("Hadi");

// clear whole Map
myMap.clear();

// get the size
myMap.size();

// Different way of loop
for (String name: myMap.keySet()){
  System.out.println(age);
}

for (String age: myMap.values()){
  System.out.println(age);
}

for(Map.Entry<String,String>entry:myMap.entrySet()){
  String key=entry.getKey();
  String value=entry.getValue();
}
```

Example:

```
class Trainer{
    private String id;
    private String name;

  public  Trainer(String id, String name)
    {
        this.id = id;
        this.name = name;
    }
    public String getID() { return id; }
    public String getName() { return name; }
}
public static void main(String[]args){

  Map<String,Trainer> train=new HashMap<>();

  train.put("red",new Trainer("40","Hadi"));
  train.put("blue",new Trainer("401","Had"));
  train.put("Yellow",new Trainer("402","Ha"));
  train.put("black",new Trainer("403","H"));

  for(Map.Entry<String,Trainer>data:train.entrySet()){

    System.out.println("Color: "+(String)data.getKey()+" id: "+(String)data.getValue().getID());
  }

}
}
```

Example:

```
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

class ExampleMap
{
   public static List<String> highEnrollmentStudents(
      Map<String, List<Course>> courseListsByStudentName, int unitThreshold)
   {
      List<String> overEnrolledStudents = new LinkedList<>();

      /*
         Build a list of the names of students currently enrolled
         in a number of units strictly greater than the unitThreshold.
      */



      for(String name:courseListsByStudentName.keySet()){
         int sum=0;
         for(Course course:courseListsByStudentName.get(name)){
            sum+=course.getNumUnits();
         }
         // int unit=courseListsByStudentName.get(name).getNumUnits();
         if (sum>unitThreshold){
            overEnrolledStudents.add(name);
         }

      }

      return overEnrolledStudents;
   }
}
```

Overriding:

toString():

```
  public String toString(){ return name;}
```

equals:

```
public boolean equals(Object o){
  if (o==null){return false;}
  if (o.getClass()!=this.getClass()){return false;}
  Theater t=(Theater)o;
  return t.seatingCapacity==seatingCapacity && t.numberTicket==numberTicket && t.name.eqals(name);
}
```

Upcasting and Downcasting:

Will it compile:

- **Check the static types,** do they all match:
  - Are you only calling the methods of the static type?
  - Are you only passing in parameters of(or lower thatn) the expected static type.(You can upcast here, but not implicity downcast)
  - Is the left side >= the right?
- Will it maybe crash:
  - Did you have to explicityly downcast to call a method.
  - Will it mybe work, but you don't know for sure becuase you didn't check?