**Lab 7**

1) Inside the Employee class of lab 3.1, override the equals method. Test it thoroughly by having equals returning true with two Employee objects that are equal, two Employee objects that are Not equal, and, …

|  |
| --- |
| @Override  **public** **boolean** equals(Object obj) {  **if** (**this** == obj) {  **return** **true**;  }  **if** (obj == **null** || getClass() != obj.getClass()) {  **return** **false**;  }  Employee other = (Employee) obj;  **return** Objects.*equals*(employeeId, other.employeeId)  && Objects.*equals*(firstName, other.firstName)  && Objects.*equals*(middleName, other.middleName)  && Objects.*equals*(lastName, other.lastName)  && Objects.*equals*(birthDate, other.birthDate)  && Objects.*equals*(SSN, other.SSN)  && Objects.*equals*(salary, other.salary);  } |

2) Inside the Position class of lab 3.1, override the equals method. Test it thoroughly by having equals returning true with two Position objects that are equal, two Position objects that are Not equal, and, …

|  |
| --- |
| @Override  **public** **boolean** equals(Object obj) {  **if** (**this** == obj) {  **return** **true**;  }  **if** (obj == **null** || getClass() != obj.getClass()) {  **return** **false**;  }  Position other = (Position) obj;  **return** Objects.*equals*(department, other.department)  && Objects.*equals*(description, other.description)  && Objects.*equals*(employee, other.employee)  && isManager == other.isManager  && Objects.*equals*(title, other.title);  } |

3) Inside the Department class of lab 3.1, override the equals method. Test it thoroughly by having equals returning true with two Department objects that are equal, two Department objects that are Not equal, and, …

|  |
| --- |
| @Override  **public** **boolean** equals(Object obj) {  **if** (**this** == obj) {  **return** **true**;  }  **if** (obj == **null** || getClass() != obj.getClass()) {  **return** **false**;  }  Department other = (Department) obj;  **return** Objects.*equals*(company.getName(), other.company.getName()) && Objects.*equals*(name, other.name);  } |

4) From lab 3.1, write a toString() method for the Employee class. Test it thoroughly!

|  |
| --- |
| @Override  **public** String toString() {  StringBuilder result = **new** StringBuilder();  result.append("Employee: ")  .append("employeeId=")  .append(employeeId)  .append(", firstName=")  .append(firstName)  .append(", middleName=")  .append(middleName)  .append(", lastName=")  .append(lastName)  .append(", birthDate=")  .append(Utils.*fomatDateTime*(birthDate))  .append(", salary=")  .append(Utils.*getPrice*(salary));  **return** result.toString();  } |

5) From lab 3.1, write a toString() method for the Position class. Test it thoroughly!

|  |
| --- |
| @Override  **public** String toString() {  StringBuilder result = **new** StringBuilder();  result.append("Position: ")  .append(" title=")  .append(title)  .append(", description=")  .append(description)  .append(", employee=\n ")  .append(employee)  .append(", department=")  .append(department);  **return** result.toString();  } |

6) From lab 3.1, write a toString() method for the Department class. Test it thoroughly!

|  |
| --- |
| @Override  **public** String toString() {  StringBuilder result = **new** StringBuilder();  result.append("Department: ")  .append(" name=")  .append(name)  .append(", location=")  .append(location)  .append(", positions=");  **if** (positions != **null** && positions.size() > 0) {  positions.stream().forEach(p -> result.append("\n ").append(p));  } **else** {  result.append("[]");  }  **return** result.toString();  } |

7) From lab 3.1, write a hashCode() method for the Employee class. Test it thoroughly by making sure that hashcode returns the same value when two Employee objects are the same (‘equals’ method should return true in this case). Different hashcodes will most ‘probably’ be created when two Employee objects are different.

|  |
| --- |
| @Override  **public** **int** hashCode() {  **return** Objects.*hash*(employeeId, firstName, middleName, lastName, birthDate, SSN, salary);  } |

8) From lab 3.1, write a hashCode() method for the Position class. Test it thoroughly by making sure that hashcode returns the same value when two Position objects are the same. Different hashcodes will most ‘probably’ be created when two Position objects are different.

|  |
| --- |
| @Override  **public** **int** hashCode() {  **return** Objects.*hash*(department, description, employee, inferiors, isManager, superior, title);  } |

**Level 3 :**

12) Inside the Position class of lab 3.1, override the clone method. Test it thoroughly,

|  |
| --- |
| @Override  **public** Object clone() **throws** CloneNotSupportedException {  Position clone = (Position) **super**.clone();  clone.setTitle(title);  clone.setDescription(description);  clone.setIsManager(isManager);  clone.setSuperior(superior);  clone.setInferiors(inferiors);  clone.setDepartment(department);  clone.setEmployee(employee);  **return** clone;  } |

13) Inside the Employee class of lab 3.1, override the clone method. Test it thoroughly,

|  |
| --- |
| @Override  **public** Object clone() **throws** CloneNotSupportedException {  Employee clone = (Employee) **super**.clone();  clone.setEmployeeId(employeeId);  clone.setFirstName(firstName);  clone.setMiddleName(middleName);  clone.setLastName(lastName);  clone.setBirthDate(birthDate);  clone.setSSN(SSN);  clone.setSalary(salary);  clone.setPosition(position);  **return** clone;  } |

**Level 4 :**

9) Create an interface with default and static methods in it. Use an IPerson interface in the first School Lab. Create a default method called myDefault that prints out “From default method.”, and, create a static method called myStatic that prints out “From static method.”. Test it all thoroughly.

Also create an abstract method called myAbstract that you must override.

|  |
| --- |
| **public** **interface** IPerson {  **void** myDefault();  **void** myStatic();  **void** myAbstract();  }  **public** **abstract** **class** Person **implements** IPerson {    …  @Override  **public** **void** myDefault() {  System.***out***.println("From default method.");  }  @Override  **public** **void** myStatic() {  System.***out***.println("From static method.");  }    @Override  **public** **void** myAbstract() {  System.***out***.println("From abstract method.");  }  } |

10) Create an enum. Use … Test it thoroughly.

|  |
| --- |
| **import** java.util.Arrays;  **public** **class** EnumsTest {  **public** **enum** ZoneTypeEnum {  ***ZONE\_1***("IA", "MT", "OR", "CA"),  ***ZONE\_2***("TX", "UT"),  ***ZONE\_3***("FL", "MA", "OH"),  ***OTHERS***();  **private** **final** Object[] zones;    ZoneTypeEnum(Object... zones) {  **this**.zones = zones;  }    **public** Object[] getZones() {  **return** **this**.zones;  }    **public** **static** ZoneTypeEnum getZoneType(String zone) {  **return** Arrays.*asList*(*values*()).stream()  .filter(t -> Arrays.*asList*(t.getZones()).contains(zone))  .findFirst()  .orElse(***OTHERS***);  }  }    **public** **static** **double** getRate(String zone) {  **double** rate = 0;  **switch** (ZoneTypeEnum.*getZoneType*(zone)) {  **case** ***ZONE\_1***:  rate = 0.35;  **break**;  **case** ***ZONE\_2***:  rate = 0.3;  **break**;  **case** ***ZONE\_3***:  rate = 0.55;  **break**;  **default**:  rate = 0.43;  }  **return** rate;  }    **public** **static** **void** main(String[] args) {  System.***out***.println(String.*format*("Rate of IA zone %s", EnumsTest.*getRate*("IA")));  System.***out***.println(String.*format*("Rate of TX zone %s", EnumsTest.*getRate*("TX")));  System.***out***.println(String.*format*("Rate of FL zone %s", EnumsTest.*getRate*("FL")));  System.***out***.println(String.*format*("Rate of NW zone %s", EnumsTest.*getRate*("NW")));  }  } |

11) Create a class that is immutable. Use … Test it thoroughly.

|  |
| --- |
| **import** java.util.HashMap;  **import** java.util.Map;  **public** **final** **class** ImmutableClass {  **private** **final** **int** id;  **private** **final** String name;  **private** **final** Map<String,String> data;    **public** ImmutableClass(**int** id, String name, Map<String,String> data) {  **super**();  **this**.id = id;  **this**.name = name;    Map<String, String> newData = **new** HashMap<>();  **for** (String key : data.keySet()) {  newData.put(key, data.get(key));  }  **this**.data = newData;  }  **public** **int** getId() {  **return** id;  }  **public** String getName() {  **return** name;  }    **public** Map<String, String> getData() {  **return** data;  }  @Override  **public** String toString() {  **return** "ImmutableClass [id=" + id + ", name=" + name + ", data=" + data + "]";  }  **public** **static** **void** main(String[] args) {  **int** id = 1;  String name = "original";  Map<String, String> data = **new** HashMap<>();  data.put("1", "first");  data.put("2", "second");    ImmutableClass immutableClass = **new** ImmutableClass(id, name, data);  System.***out***.println("ImmutableClass before changed: " + immutableClass);  System.***out***.println(data == immutableClass.getData());  // Change values  id = 2;  name = "changed";  data.put("3", "third");  System.***out***.println("The map data has been changed: " + data);  System.***out***.println("ImmutableClass after changed: " + immutableClass);    }  } |