1) Write an abstract class called **Room** with two attributes, String building, the building in which the room is located, and int number, the room number.

```java
public abstract class Room {
  protected String building;
  protected int number;

  public Room(String building, int number) {
    this.building = building;
    this.number = number;
  }
}
```

2) Write an interface called **Securable**, which requires the coding of a method, boolean hasAccess(String id), removeAccess(String id), addAccess(String id), which each take a String id as argument, representing a person's id code

```java
public interface Securable {
  public boolean hasAccess(String id);
  public void removeAccess(String id);
  public void addAccess(String id);
}
```

3) Write an abstract class called **StudyRoom** which inherits from **Room** and implements **Securable** and **Comparable**. StudyRooms are considered equal if the building is the same. Otherwise, if the buildings are different, then the comparison is done on the room number.

```java
import java.util.ArrayList;

public abstract class StudyRoom extends Room implements Securable,
Comparable<StudyRoom> {
  private ArrayList<String> access_id = new ArrayList<>();
```

```java
public StudyRoom(String building, int number) {
  super(building, number);
}

public StudyRoom(String building, int number, ArrayList<String> access_id) {
  super(building, number);
  this.access_id = access_id;
}

@Override
public boolean hasAccess(String id) {
  if (access_id.contains(id)) {
    return true;
  }
  return false;
}

@Override
public void removeAccess(String id) {
  if (access_id.contains(id)) {
    access_id.remove(access_id.indexOf(id));
  }
  else {
    System.out.println("This id doesn't have access.");
  }
}

@Override
public void addAccess(String id) {
  if (access_id.contains(id)) {
    System.out.println("This id has access already.");
  }
  else {
    access_id.add(id);
  }
}

@Override
public int compareTo(StudyRoom a) {
  if (a.building.toLowerCase().equals(this.building.toLowerCase())) {
    return 0;
  }
  else {
    return this.number - a.number;
```

```
    }
  }
}
```

4) Implement a class called **IsenbergStudyRoom** which inherits from **StudyRoom**, and implements hasAcess, removeAccess, and addAccess.
   hasAccess() checks if the id is present in the object.
   removeAccess() removes the id from the object if it is present
   addAccess() adds an id only if the id is not already present in the object
   (hint: use a list as an attribute of the class to store IDs)

Java

```java
import java.util.ArrayList;

public class IsenbergStudyRoom extends StudyRoom{

  public IsenbergStudyRoom(String building, int number) {
    super(building, number);
  }

  public IsenbergStudyRoom(String building, int number, ArrayList<String>
access_id) {
    super(building, number, access_id);
  }

  public boolean hasAccess(String id) {
    return super.hasAccess(id);
  }

  public void removeAccess(String id) {
    super.removeAccess(id);
  }

  public void addAccess(String id) {
    super.addAccess(id);
  }
}
```

5) Create an abstract class called **Product** with two attributes: productName (of type String) and price (of type double).

- Write an interface named Discountable that requires the implementation of methods:
  - findDiscount(double discountPercentage) to find a discount to the product's price.
  - resetPrice() to reset the product's price to its discounted value.
- Develop an abstract class named **ElectronicProduct** that inherits from **Product** and implements Discountable.

6) Write an abstract class called **Vegetation**, with a single attribute called **name** (of type String). Class **Vegetation** implements two methods: **getName()**, which returns a **String**; and **setName(String s)**, which does not return a value. Class **Vegetation** has an abstract method called **producesFruit()** that returns a boolean value.

7) Write an interface called **Localizable**, which requires the coding of a method called **getLocation()**, which returns an **ArrayList** of **Integer** values.

```java
import java.util.ArrayList;

public interface Localizable {
  ArrayList<Integer> getLocation();
}
```

8) Write an abstract class called **Tree** which inherits from **Vegetation**, and which implements both **Localizable** and **Comparable**. Class **Tree** has an attribute of type float that stores its current height. The comparison of two objects of class **Tree** is based on their relative height.

```java
import java.util.ArrayList;
```

```
public abstract class Tree extends Vegetation implements Localizable,
Comparable<Tree> {
  private float height;
  // Constructor
  public Tree(String name, float height) {
    super(name);
    this.height = height;
  }
  // Getter for height
  public float getHeight() {
    return height;
  }
  // Setter for height
  public void setHeight(float height) {
    this.height = height;
  }
  // Implementation of producesFruit method from Vegetation
  @Override
  public abstract boolean producesFruit();
  // Implementation of getLocation method from Localizable
  @Override
  public ArrayList<Integer> getLocation() {
    // You can provide the logic to get the location of the tree here
    // For example, you might return an ArrayList of coordinates.
    ArrayList<Integer> location = new ArrayList<>();
    // Add your location data here, e.g., latitude and longitude
    location.add(0); // Example: Latitude
    location.add(0); // Example: Longitude
    return location;
  }
  // Implementation of compareTo method from Comparable
  @Override
  public int compareTo(Tree otherTree) {
    // Compare trees based on their relative height
    if (this.height < otherTree.height) {
      return -1;
    } else if (this.height > otherTree.height) {
      return 1;
    } else {
      return 0;
```

```
    }
  }
}
```

9) Implement a class called **OakTree**, which inherits from **Tree**. **OakTree.getLocation()** returns an **ArrayList** with the values [7,8,9,10]. **OakTree.producesFruit()** returns **true**. 1

Java

```java
import java.util.ArrayList;
public class OakTree extends Tree {
  // Constructor for OakTree
  public OakTree(String name, float height) {
    super(name, height);
  }
  // Override producesFruit method to return true for OakTree
  @Override
  public boolean producesFruit() {
    return true;
  }
  // Override getLocation method to return specific location for OakTree
  @Override
  public ArrayList<Integer> getLocation() {
    ArrayList<Integer> location = new ArrayList<>();
    location.add(7); // Latitude
    location.add(8); // Longitude
    location.add(9); // Altitude
    location.add(10); // Depth
    return location;
  }
  // You can add additional methods or properties specific to OakTree
here
}
```

.

10) Write an abstract class called **FoodItem**, with the following attributes: an attribute called **name,** of type String; an attribute called servingSize, of type double; and an attribute called

calories, of type double. Class **FoodItem** implements get and set methods for all of its three attributes. Class **FoodItem** has an abstract method called **dilutable()** that returns a boolean value; Class FoodItem implements the Comparable interface, with comparisons based on the ratio between calories and servingSize (i.e. calories/servingSize).

Java

```java
public abstract class FoodItem implements Comparable<FoodItem> {
   private String name;
   private double servingSize;
   private double calories;

   public FoodItem(String name, double servingSize, double calories) {
       this.name = name;
       this.servingSize = servingSize;
       this.calories = calories;
   }

   public String getName() {
       return name;
   }
   public void setName(String name) {
       this.name = name;
   }

   public double getServingSize() {
       return servingSize;
   }
   public void setServingSize(double servingSize) {
       this.servingSize = servingSize;
   }

   public double getCalories() {
       return calories;
   }
   public void setCalories(double calories) {
       this.calories = calories;
   }

   public abstract Boolean dilutable();

   @Override
   public int compareTo(FoodItem otherFoodItem) {
```

```
        if ((this.calories/this.servingSize) <
(otherFoodItem.getCalories()/otherFoodItem.getServingSize())) {
            return -1;
        } else if ((this.calories/this.servingSize) >
(otherFoodItem.getCalories()/otherFoodItem.getServingSize())){
            return 1;
        } else {
            return 0;
        }
    }
}
```

11) Write an interface called **Energetic**, which requires the coding of a method called **workCapacity()**, which returns a double.

Java

```
public interface Energetic {
    public double workCapacity();
}
```

12) Write a class called **Breakfast** which inherits from **FoodItem.** Class **Breakfast** has an attribute called carbohydrates of type double and an attribute called protein of type double. **Breakfast.workCapacity()** returns the object's calorie attribute. **Breakfast.dilutable()** returns false.

Java

```
public class Breakfast extends FoodItem implements Energetic {
    private double carbohydrates;
    private double protein;
    public Breakfast(String name, double servingSize, double calories, double
carbohydrates, double protein) {
        super(name, servingSize, calories);
        this.carbohydrates = carbohydrates;
```

```
        this.protein = protein;
    }

    public double getCarbohydrates() {
        return carbohydrates;
    }
    public void setCarbohydrates(double carbohydrates) {
        this.carbohydrates = carbohydrates;
    }
    public double getProtein() {
        return protein;
    }
    public void setProtein(double protein) {
        this.protein = protein;
    }
    public Boolean dilutable() {
        return false;
    }

    public double workCapacity() {
        return getCalories();
    }
}
```

13) Implement a class called **Oatmeal**, which inherits from **Breakfast**.

Java

```
public class Oatmeal extends Breakfast {
  public Oatmeal(String name, double servingSize, double calories, double
carbohydrates, double protein) {
    super(name, servingSize, calories, carbohydrates, protein);
  }
}
```