1.

```
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;
public class LoadIrisDataset {
    public static void main(String[] args) {
        try {
            DataSource source = new DataSource("iris.arff");
            Instances dataset = source.getDataSet();
            System.out.println("Dataset Relation Name: " + dataset.relationName());
            System.out.println("Number of Instances: " + dataset.numInstances());
            System.out.println("Number of Attributes: " + dataset.numAttributes());
            if (dataset.classIndex() == -1)
                dataset.setClassIndex(dataset.numAttributes() - 1);
            System.out.println("Class Attribute Name: " +
dataset.classAttribute().name());
            System.out.println("\nFirst 10 Instances:");
            for (int i = 0; i < 10 && i < dataset.numInstances(); i++) {
                System.out.println((i + 1) + ": " + dataset.instance(i));
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2.

```
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.trees.J48;
import weka.classifiers.Evaluation;
import java.util.Random;
public class J48IrisClassifier {
    public static void main(String[] args) {
        try {
            DataSource source = new DataSource("iris.arff");
            Instances dataset = source.getDataSet();
            if (dataset.classIndex() == -1) {
                dataset.setClassIndex(dataset.numAttributes() - 1);
            }
            J48 tree = new J48();
            tree.buildClassifier(dataset);
            Evaluation eval = new Evaluation(dataset);
            eval.crossValidateModel(tree, dataset, 10, new Random(1));
            double accuracy = (1 - eval.errorRate()) * 100;
            System.out.println("Accuracy: " + String.format("%.2f", accuracy) + "%");
            double[][] cmMatrix = eval.confusionMatrix();
            System.out.println("\nConfusion Matrix:");
            for (int i = 0; i < cmMatrix.length; i++) {
                for (int j = 0; j < cmMatrix[i].length; j++) {
                    System.out.print((int) cmMatrix[i][j] + "\t");
                }
                System.out.println();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

3.

```
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.trees.J48;
import weka.classifiers.Evaluation;

import java.util.Random;
public class J48TrainTestSplit {
    public static void main(String[] args) {
        try {
            DataSource source = new DataSource("iris.arff");
            Instances dataset = source.getDataSet();
            if (dataset.classIndex() == -1) {
                dataset.setClassIndex(dataset.numAttributes() - 1);
            }
            dataset.randomize(new Random(1));
            int trainSize = (int) Math.round(dataset.numInstances() * 0.7);
            int testSize = dataset.numInstances() - trainSize;

            Instances trainSet = new Instances(dataset, 0, trainSize);
            Instances testSet = new Instances(dataset, trainSize, testSize);
            J48 tree = new J48();
            tree.buildClassifier(trainSet);
            Evaluation eval = new Evaluation(trainSet);
            eval.evaluateModel(tree, testSet);
            System.out.println("Training set size: " + trainSet.numInstances());
            System.out.println("Test set size: " + testSet.numInstances());
            double accuracy = (1 - eval.errorRate()) * 100;
            System.out.println("Accuracy on test set: " + String.format("%.2f", accuracy) + "%");
            double[][] cmMatrix = eval.confusionMatrix();
            System.out.println("\nConfusion Matrix:");
            for (int i = 0; i < cmMatrix.length; i++) {
                for (int j = 0; j < cmMatrix[i].length; j++) {
                    System.out.print((int) cmMatrix[i][j] + "\t");
                }
                System.out.println();
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4.

```
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.Discretize;
import weka.filters.unsupervised.attribute.Remove;

public class PreprocessIris {
    public static void main(String[] args) {
        try {
            // Load dataset
            DataSource source = new DataSource("iris.arff");
            Instances dataset = source.getDataSet();

            // Print original dataset structure
            System.out.println("=== Original Dataset Structure ===");
            System.out.println(dataset);

            /* ------------------ Step 1: Discretize numeric attributes ------------------ */
            Discretize discretize = new Discretize();
            discretize.setBins(5); // 5 bins
            discretize.setInputFormat(dataset);
            Instances discretizedData = Filter.useFilter(dataset, discretize);

            /* ------------------ Step 2: Remove the first attribute ------------------ */
            Remove remove = new Remove();
            remove.setAttributeIndices("1"); // remove first attribute (index starts
from 1 in Weka)
            remove.setInputFormat(discretizedData);
            Instances finalData = Filter.useFilter(discretizedData, remove);

            // Print dataset structure after preprocessing
            System.out.println("\n=== Dataset After Preprocessing (Discretization + 
Remove First Attribute) ===");
            System.out.println(finalData);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

5.

```
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.bayes.NaiveBayes;
import weka.core.SerializationHelper;

public class NaiveBayesIris {
    public static void main(String[] args) {
        try {
            DataSource source = new DataSource("iris.arff");
            Instances dataset = source.getDataSet();

            if (dataset.classIndex() == -1) {
                dataset.setClassIndex(dataset.numAttributes() - 1);
            }

            NaiveBayes nb = new NaiveBayes();
            nb.buildClassifier(dataset);

            SerializationHelper.write("naivebayes.model", nb);
            System.out.println("NaiveBayes model trained and saved successfully!");

            NaiveBayes loadedModel = (NaiveBayes)
SerializationHelper.read("naivebayes.model");

            double labelIndex = loadedModel.classifyInstance(dataset.instance(0));
            String predictedClass = dataset.classAttribute().value((int) labelIndex);

            System.out.println("\nFirst Instance: " + dataset.instance(0));
            System.out.println("Predicted Class Label: " + predictedClass);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

6.

```
import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;
import weka.clusterers.SimpleKMeans;

public class KMeansIris {
    public static void main(String[] args) {
        try {
}
      DataSource source = new DataSource("iris.arff");
      Instances dataset = source.getDataSet();



      SimpleKMeans kmeans = new SimpleKMeans();
      kmeans.setNumClusters(3); // 3 clusters
      kmeans.setSeed(10);       // For reproducibility
      kmeans.setPreserveInstancesOrder(true); // Keep order for assignments
      kmeans.buildClusterer(dataset);
      System.out.println("=== Cluster Centroids ===");
      Instances centroids = kmeans.getClusterCentroids();
      for (int i = 0; i < centroids.numInstances(); i++) {
        System.out.println("Cluster " + i + ": " + centroids.instance(i));
      }
      System.out.println("\n=== Cluster Assignments ===");
      int[] assignments = kmeans.getAssignments();
      for (int i = 0; i < assignments.length; i++) {
        System.out.println("Instance " + (i + 1) + " -> Cluster " +
assignments[i]);
      }

    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

7.

```java
import weka.core.Instances;

import weka.core.converters.ConverterUtils.DataSource;


public class CustomSepalLengthBins {

    public static String convertSepalLength(double value) {

        if (value < 5.0) {

            return "Short";

        } else if (value <= 6.5) {

            return "Medium";

        } else {

            return "Tall";

        }

    }


    public static void main(String[] args) {

        try {

            DataSource source = new DataSource("iris.arff");

            Instances dataset = source.getDataSet();

            if (dataset.classIndex() == -1) {

                dataset.setClassIndex(dataset.numAttributes() - 1);

            }

            System.out.println("=== Iris Dataset with Nominal Sepal Length ===");

            for (int i = 0; i < dataset.numInstances(); i++) {

                double sepalLength = dataset.instance(i).value(0); // First attribute

                String nominalLength = convertSepalLength(sepalLength);
                System.out.println("Instance " + (i + 1) + ": "
                        + "SepalLength=" + sepalLength
                        + " (" + nominalLength + "), "
                        + "Other Attributes=" + dataset.instance(i).toString());
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```