Generating Random Text

Interfaces and Abstract Classes

Duke
UNIVERSITY

Commonalities in Code

- Method `MarkovRunner.runMarkov`

```
public void runMarkov() {
  FileResource fr = new FileResource();
  String st = fr.asString();
  st = st.replace('\n', ' ');
  MarkovZero markov = new MarkovZero();
  markov.setTraining(st);
  for(int k=0; k < 3; k++){
    String text = markov.getRandomText(200);
    printOut(text);
  }
}
```

Commonalities in Code

- Method `MarkovRunner.runMarkov`
  - We changed MarkovZero to MarkovOne

```
public void runMarkov() {
  FileResource fr = new FileResource();
  String st = fr.asString();
  st = st.replace('\n', ' ');
  MarkovZero markov = new MarkovZero();
  markov.setTraining(st);
  for(int k=0; k < 3; k++){
    String text = markov.getRandomText(200);
    printOut(text);
  }
}
```

Commonalities in Code

- Method `MarkovRunner.runMarkov`
  - We changed MarkovZero to MarkovOne

```
public void runMarkov() {
  FileResource fr = new FileResource();
  String st = fr.asString();
  st = st.replace('\n', ' ');
  MarkovOne markov = new MarkovOne();
  markov.setTraining(st);
  for(int k=0; k < 3; k++){
    String text = markov.getRandomText(200);
    printOut(text);
  }
}
```

Commonalities in Code

- Method `MarkovRunner.runMarkov`
  - We changed MarkovZero to MarkovOne
  - Code still worked! Why?

```
public void runMarkov() {
  FileResource fr = new FileResource();
  String st = fr.asString();
  st = st.replace('\n', ' ');
  MarkovOne markov = new MarkovOne();
  markov.setTraining(st);
  for(int k=0; k < 3; k++){
    String text = markov.getRandomText(200);
    printOut(text);
  }
}
```

Commonalities in Code

- Method `MarkovRunner.runMarkov`
  - We changed MarkovZero to MarkovOne
  - Code still worked! Why?

```
public void runMarkov() {
  FileResource fr = new FileResource();
  String st = fr.asString();
  st = st.replace('\n', ' ');
  MarkovOne markov = new MarkovOne();
  markov.setTraining(st);
  for(int k=0; k < 3; k++){
    String text = markov.getRandomText(200);
    printOut(text);
  }
}
```

Commonalities in Code

- Method `MarkovRunner.runMarkov`
  - We changed MarkovZero to MarkovOne
  - Code still worked! Why?

```
public void runMarkov() {
  FileResource fr = new FileResource();
  String st = fr.asString();
  st = st.replace('\n', ' ');
  MarkovOne markov = new MarkovOne();
  markov.setTraining(st);
  for(int k=0; k < 3; k++){
    String text = markov.getRandomText(200);
    printOut(text);
  }
}
```

Commonalities in Code

- Method `MarkovRunner.runMarkov`
  - We changed MarkovZero to MarkovOne
  - Code still worked! Why?
- Capture code commonalities using a Java Interface
  - Just as you did with Comparable and Comparator
  - Methods `setTraining()` and `getRandomText(int)`

Developing an Interface

- Common method signatures in Interface

```
public interface IMarkovModel {
  public void setTraining(String text);
  public String getRandomText(int numChars);
}
```

Developing an Interface

- Common method signatures in Interface

```
public interface IMarkovModel {
  public void setTraining(String text);
  public String getRandomText(int numChars);
}
```

Developing an Interface

- Common method signatures in Interface
  - Name interfaces starting with "I" is common; we create IMarkovModel with required methods

```
public interface IMarkovModel {
  public void setTraining(String text);
  public String getRandomText(int numChars);
}
```

Developing an Interface

- Common method signatures in Interface
  - Name interfaces starting with "I" is common; we create IMarkovModel with required methods
  - MarkovZero, MarkovOne, MarkovTwo—each implement interface; methods already there!

```
public class MarkovOne implements IMarkovModel{

  private String myText;
  private Random myRandom;

  public MarkovOne() {
    myRandom = new Random();
  }
```

Developing an Interface

- Common method signatures in Interface
  - Name interfaces starting with "I" is common; we create IMarkovModel with required methods
  - MarkovZero, MarkovOne, MarkovTwo—each implement interface; methods already there!

```
public class MarkovTwo implements IMarkovModel{

  private String myText;
  private Random myRandom;

  public MarkovTwo() {
    myRandom = new Random();
  }
```

Interface: Utility and Flexibility

- Write methods that use Interface

```
public void runModel(IMarkovModel markov,
                     String text, int size){
    markov.setTraining(text);
    System.out.println("running with "+markov);
    for(int k=0; k < 3; k++){
      String st = markov.getRandomText(size);
      printOut(st);
    }
}
```

Interface: Utility and Flexibility

- Write methods that use Interface

```
public void runModel(IMarkovModel markov,
                     String text, int size){
    markov.setTraining(text);
    System.out.println("running with "+markov);
    for(int k=0; k < 3; k++){
      String st = markov.getRandomText(size);
      printOut(st);
    }
}
```

Interface: Utility and Flexibility

- Write methods that use Interface

```
public void runModel(IMarkovModel markov,
                     String text, int size){
    markov.setTraining(text);
    System.out.println("running with "+markov);
    for(int k=0; k < 3; k++){
      String st = markov.getRandomText(size);
      printOut(st);
    }
}
```

```
MarkovZero mz = new MarkovZero();
runModel(mz,text,800);
```

Interface: Utility and Flexibility

- Write methods that use Interface

```
public void runModel(IMarkovModel markov,
                     String text, int size){
    markov.setTraining(text);
    System.out.println("running with "+markov);
    for(int k=0; k < 3; k++){
      String st = markov.getRandomText(size);
      printOut(st);
    }
}
```

```
MarkovZero mz = new MarkovZero();
runModel(mz,text,800);
```

```
MarkovTwo m2 = new MarkovTwo();
runModel(m2,text,800);
```

Interface: Utility and Flexibility

- Write methods that use Interface

```
public void runModel(IMarkovModel markov,
                     String text, int size){
    markov.setTraining(text);
    System.out.println("running with "+markov);
    for(int k=0; k < 3; k++){
      String st = markov.getRandomText(size);
      printOut(st);
    }
}
```

```
MarkovZero mz = new MarkovZero();
runModel(mz,text,800);
```

```
MarkovTwo m2 = new MarkovTwo();
runModel(m2,text,800);
```

Interface: Utility and Flexibility

- Write methods that use Interface

```
public void runModel(IMarkovModel markov,
                     String text, int size){
    markov.setTraining(text);
    System.out.println("running with "+markov);
    for(int k=0; k < 3; k++){
      String st = markov.getRandomText(size);
      printOut(st);
    }
}
```

```
MarkovZero mz = new MarkovZero();
runModel(mz,text,800);
```

```
MarkovTwo m2 = new MarkovTwo();
runModel(m2,text,800);
```

Interface: Utility and Flexibility

- Write methods that use Interface

```
public void runModel(IMarkovModel markov,
                     String text, int size){
    markov.setTraining(text);
    System.out.println("running with "+markov);
    for(int k=0; k < 3; k++){
      String st = markov.getRandomText(size);
      printOut(st);
    }
}
```

```
MarkovZero mz = new MarkovZero();
runModel(mz,text,800);
```

```
MarkovTwo m2 = new MarkovTwo();
runModel(m2,text,800);
```

Interface: Utility and Flexibility

- Write methods that use Interface

```
public void runModel(IMarkovModel markov,
                     String text, int size){
    markov.setTraining(text);
    System.out.println("running with "+markov);
    for(int k=0; k < 3; k++){
      String st = markov.getRandomText(size);
      printOut(st);
    }
}
```

```
MarkovZero mz = new MarkovZero();
runModel(mz,text,800);
```

```
MarkovTwo m2 = new MarkovTwo();
runModel(m2,text,800);
```

Software Design: Open-Closed

- **Open** for extension, **closed** to modification
  - Don't change tested/proven code!
- Interface provides flexibility
  - Add new, general MarkovModel class
  - All orders 1,2,… implements **`IMarkovModel`**
  - No change to existing code! e.g., **`runModel`**
- Implement efficient version with HashMap
  - Avoid re-scanning for 'th' follows
  - Still use existing code with no change!

Abstract Class

- MarkovOne, MarkovTwo, MarkovModel classes share state and code
  - Each class has random number generator and text in instance variables **`myRandom`**, **`myText`**
  - Duplicated helper method **`getFollows(key)`**
- Capture commonality in Abstract Base Class
  - Relies on object-oriented concept: **Inheritance**
  - Used extensively in java.util: AbstractList, and AbstractMap

AbstractMarkovModel

- Class marked as abstract

```
public abstract class AbstractMarkovModel
        implements IMarkovModel {

  protected  String myText;
  protected  Random myRandom;

  public AbstractMarkovModel() {
    myRandom = new Random();
  }

  public void setTraining(String text) {
    myText = text;
  }
```

AbstractMarkovModel

- Class marked as abstract

```
public abstract class AbstractMarkovModel
        implements IMarkovModel {

  protected  String myText;
  protected  Random myRandom;

  public AbstractMarkovModel() {
    myRandom = new Random();
  }

  public void setTraining(String text) {
    myText = text;
  }
```

AbstractMarkovModel

- Class marked as abstract
  - Shared state is protected, not private

```
public abstract class AbstractMarkovModel
        implements IMarkovModel {

  protected String myText;
  protected Random myRandom;

  public AbstractMarkovModel() {
    myRandom = new Random();
  }

  public void setTraining(String text) {
    myText = text;
  }
```

AbstractMarkovModel

- Class marked as abstract
  - Shared state is protected, not private

```
public abstract class AbstractMarkovModel
        implements IMarkovModel {

  protected  String myText;
  protected  Random myRandom;

  public AbstractMarkovModel() {
    myRandom = new Random();
  }

  public void setTraining(String text) {
    myText = text;
  }
```

Abstract and Shared Methods

- At least one method marked as abstract

```
public abstract class AbstractMarkovModel
        implements IMarkovModel {

 // state and shared methods here

  abstract public String getRandomText(int numChars);

  protected ArrayList<String> getFollows(String key){
      … // code not shown
   }
```

Abstract and Shared Methods

- At least one method marked as abstract

```
public abstract class AbstractMarkovModel
        implements IMarkovModel {

 // state and shared methods here

  abstract public String getRandomText(int numChars);

  protected ArrayList<String> getFollows(String key){
      … // code not shown
   }
```

Abstract and Shared Methods

- At least one method marked as abstract
  - Must be implemented in subclasses

```
public abstract class AbstractMarkovModel
       implements IMarkovModel {

 // state and shared methods here

  abstract public String getRandomText(int numChars);

  protected ArrayList<String> getFollows(String key){
      … // code not shown
   }
```

Abstract and Shared Methods

- At least one method marked as abstract
  - Must be implemented in subclasses
- Shared helper functions protected

```
public abstract class AbstractMarkovModel
        implements IMarkovModel {

 // state and shared methods here

  abstract public String getRandomText(int numChars);

  protected ArrayList<String> getFollows(String key){
      … // code not shown
   }
```

Extending the Base Class

- When ***base*** class extended, get access to protected instance variables and methods from ***super*** class

```
public class MarkovModel extends AbstractMarkovModel {

  private int myOrder;

  public MarkovModel(int order) {
    myOrder = order;
  }
  // more code here
```

Extending the Base Class

- When ***base*** class extended, get access to protected instance variables and methods from ***super*** class

```
public class MarkovModel extends AbstractMarkovModel {

  private int myOrder;

  public MarkovModel(int order) {
    myOrder = order;
  }
  // more code here
```

Extending the Base Class

- When ***base*** class extended, get access to protected instance variables and methods from ***super*** class
  - Also implement interfaces of base class

```
public class MarkovModel extends AbstractMarkovModel {

  private int myOrder;

  public MarkovModel(int order) {
    myOrder = order;
  }
  // more code here
```

Extending the Base Class

- When ***base*** class extended, get access to protected instance variables and methods from ***super*** class
  - Also implement interfaces of base class
  - Can have instance variables as well

```
public class MarkovModel extends AbstractMarkovModel {

  private int myOrder;

  public MarkovModel(int order) {
    myOrder = order;
  }
  // more code here
```

Implementing Abstract Methods

- Must implement abstract method(s)

```
public class MarkovModel extends AbstractMarkovModel {

  public String getRandomText(int length) {
    StringBuffer sb = new StringBuffer();
    int index = myRandom.nextInt(myText.length() - myOrder);
    String current = myText.substring(index, index + myOrder);
    sb.append(current);
    for(int k=0; k < length-myOrder; k++){
      ArrayList<String> follows = getFollows(current);
```

Implementing Abstract Methods

- Must implement abstract method(s)

```
public class MarkovModel extends AbstractMarkovModel {

  public String getRandomText(int length) {
    StringBuffer sb = new StringBuffer();
    int index = myRandom.nextInt(myText.length() - myOrder);
    String current = myText.substring(index, index + myOrder);
    sb.append(current);
    for(int k=0; k < length-myOrder; k++){
      ArrayList<String> follows = getFollows(current);
```

Implementing Abstract Methods

- Must implement abstract method(s)
- Have access to protected state/behavior

```
public class MarkovModel extends AbstractMarkovModel {

  public String getRandomText(int length) {
    StringBuffer sb = new StringBuffer();
    int index = myRandom.nextInt(myText.length() - myOrder);
    String current = myText.substring(index, index + myOrder);
    sb.append(current);
    for(int k=0; k < length-myOrder; k++){
      ArrayList<String> follows = getFollows(current);
```

Implementing Abstract Methods

- Must implement abstract method(s)
- Have access to protected state/behavior
  - **`myRandom`** and **`myText`**

```
public class MarkovModel extends AbstractMarkovModel {

  public String getRandomText(int length) {
    StringBuffer sb = new StringBuffer();
    int index = myRandom.nextInt(myText.length() - myOrder);
    String current = myText.substring(index, index + myOrder);
    sb.append(current);
    for(int k=0; k < length-myOrder; k++){
      ArrayList<String> follows = getFollows(current);
```

Implementing Abstract Methods

- Must implement abstract method(s)
- Have access to protected state/behavior
  - **`myRandom`** and **`myText`**

```
public class MarkovModel extends AbstractMarkovModel {

  public String getRandomText(int length) {
    StringBuffer sb = new StringBuffer();
    int index = myRandom.nextInt(myText.length() - myOrder);
    String current = myText.substring(index, index + myOrder);
    sb.append(current);
    for(int k=0; k < length-myOrder; k++){
      ArrayList<String> follows = getFollows(current);
```

Implementing Abstract Methods

- Must implement abstract method(s)
- Have access to protected state/behavior
  - **`myRandom`** and **`myText`**
  - **`getFollows(...)`**

```
public class MarkovModel extends AbstractMarkovModel {

  public String getRandomText(int length) {
    StringBuffer sb = new StringBuffer();
    int index = myRandom.nextInt(myText.length() - myOrder);
    String current = myText.substring(index, index + myOrder);
    sb.append(current);
    for(int k=0; k < length-myOrder; k++){
      ArrayList<String> follows = getFollows(current);
```

Implementing Abstract Base Class

- Implement the IMarkovModel interface
  - Supply default functionality when possible
  - Avoid duplicating state and code
- Subclasses extend AbstractMarkovModel
  - They implement IMarkovModel too!
  - Interface the same, client code doesn't change
- Some methods are abstract
  - `getRandomText` implementation changes