Data Abstraction
Software Development. Review of Java Constructs.

Janyl Jumadinova

September 14–16, 2020
Collections

- **Collection**: an object that stores data; a.k.a. “data structure”
- The objects stored are called **elements**
- Some collections maintain an ordering; some allow duplicates
- Typical operations: add, remove, clear, contains (search), size

Examples found in the Java class libraries: ArrayList, LinkedList, HashMap, TreeSet, PriorityQueue

All collections are in the java.util package.
Collections

- **Collection**: an object that stores data; a.k.a. “data structure”
- The objects stored are called **elements**
- Some collections maintain an ordering; some allow duplicates
- Typical operations: add, remove, clear, contains (search), size
- Examples found in the Java class libraries: `ArrayList`, `LinkedList`, `HashMap`, `TreeSet`, `PriorityQueue`
- all collections are in the `java.util` package
Collections

```
<interface> Collection
  \- <interface> List
    \- AbstractList
      \- ArrayList
        \- LinkedList
    \- AbstractCollection
    \- Set
      \- AbstractSet
      \- SortedSet
        \- HashSet
        \- TreeSet
```
Lists

- **List**: a collection storing an ordered sequence of elements
- Each element is accessible by a 0-based index
- A list has a size (number of elements that have been added)
- Elements can be added to the front, back, or elsewhere
Lists

- **List**: a collection storing an ordered sequence of elements
- Each element is accessible by a 0-based index
- A list has a size (number of elements that have been added)
- Elements can be added to the front, back, or elsewhere
- In Java, a list can be represented as an `ArrayList` object
Lists

- **List**: a collection storing an ordered sequence of elements
- Each element is accessible by a 0-based index
- A list has a size (number of elements that have been added)
- Elements can be added to the front, back, or elsewhere
- In Java, a list can be represented as an **ArrayList** object
A Few ArrayList Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(value)</td>
<td>appends value at end of list</td>
</tr>
<tr>
<td>add(index, value)</td>
<td>inserts given value just before the given index, shifting subsequent values to the right</td>
</tr>
<tr>
<td>clear()</td>
<td>removes all elements of the list</td>
</tr>
<tr>
<td>indexOf(value)</td>
<td>returns first index where given value is found in list (-1 if not found)</td>
</tr>
<tr>
<td>get(index)</td>
<td>returns the value at given index</td>
</tr>
<tr>
<td>remove(index)</td>
<td>removes/returns value at given index, shifting subsequent values to the left</td>
</tr>
<tr>
<td>set(index, value)</td>
<td>replaces value at given index with given value</td>
</tr>
<tr>
<td>size()</td>
<td>returns the number of elements in list</td>
</tr>
<tr>
<td>toString()</td>
<td>returns a string representation of the list such as &quot;[3, 42, -7, 15]&quot;</td>
</tr>
</tbody>
</table>
Type Parameters (Generics)

- `ArrayList<Type> name = new ArrayList<Type>()`;
- When constructing an `ArrayList`, you must specify the type of elements it will contain between `<` and `>`. 
- This is called a **type parameter** or a **generic class**. 
- Allows the same `ArrayList` class to store lists of different types.
Type Parameters (Generics)

- `ArrayList<Type> name = new ArrayList<Type>();`
- When constructing an `ArrayList`, you must specify the type of elements it will contain between `<` and `>`. This is called a **type parameter** or a **generic class**.
- Allows the same `ArrayList` class to store lists of different types.
  ```java
  ArrayList<String> names = new ArrayList<String>();
  names.add("Marty Stepp");
  names.add("Stuart Reges");
  ```
ArrayList of Primitives?

- The type you specify when creating an ArrayList must be an object type; it cannot be a primitive type.
The type you specify when creating an ArrayList must be an object type; it cannot be a primitive type.

// illegal -- int cannot be a type parameter
ArrayList<int> list = new ArrayList<int>();
The type you specify when creating an ArrayList must be an object type; it cannot be a primitive type.

```java
// illegal -- int cannot be a type parameter
ArrayList<int> list = new ArrayList<int>();
```

But we can still use ArrayList with primitive types by using special classes called **wrapper classes** in their place.

```java
// creates a list of ints
ArrayList<Integer> list = new ArrayList<Integer>();
```
A wrapper is an object whose sole purpose is to hold a primitive value.

Once you construct the list, use it with primitives as normal:

```java
ArrayList<Double> grades = new ArrayList<Double>();
grades.add(3.2);
grades.add(2.7);
...
double myGrade = grades.get(0);
```
Random Class

- The Random class is part of the java.util package
- It provides methods that generate pseudo-random numbers
Random Class

- The Random class is part of the java.util package
- It provides methods that generate pseudo-random numbers
- Before you can use its methods, you must create an instance of the Random class
- `Random rand = new Random();`
Random Class

float nextFloat( )
float nextDouble( )
- Returns a random number between 0.0 inclusive and 1.0 exclusive.
Random Class

```java
float nextFloat( )
float nextDouble( )
- Returns a random number between 0.0 inclusive and 1.0 exclusive.
Random rand = new Random( );
float f;
f = rand.nextFloat( );
```
int nextInt() - Returns a random number that ranges over all possible int values positive and negative.
Random Class

int nextInt( ) - Returns a random number that ranges over all possible int values positive and negative.
Random rand = new Random( );
int num;
num = rand.nextInt( );
**Random Class**

```java
int nextInt( ) - Returns a random number that ranges over all possible int values positive and negative.
Random rand = new Random( );
int num;
num = rand.nextInt( );
```

```java
int nextInt( int num ) - Returns a random number between 0 (inclusive) and num (exclusive).
Random rand = new Random( );
int num;
num = rand.nextInt(5 );
```
Random Class

```java
int nextInt( ) - Returns a random number that ranges over all possible int values positive and negative.
Random rand = new Random( );
int num;
num = rand.nextInt( );
```

```java
int nextInt( int num ) - Returns a random number between 0 (inclusive) and num (exclusive).
Random rand = new Random( );
int num;
num = rand.nextInt(5 );
```
Math Class

Math class is part of the java.lang package (no need to import).

Math class consists of:

- **static methods**, which are methods that don’t depend on the contents of an object.

- **static fields**, which are values that are usually defined to be public, final and static, meaning that anyone can access them outside the package. Since their values are final, that means that they are constant and can’t be changed.
Software Development

1. Design.
2. Implementation.
3. Testing.
4. Debugging.
Design

Separation into classes, class interaction, data, actions

- **Responsibilities**: using action words.
- **Independence**: autonomy of each class.
- **Behaviors**: actions of each class.
Design

Separation into classes, class interaction, data, actions

- **Responsibilities**: using action words.
- **Independence**: autonomy of each class.
- **Behaviors**: actions of each class.

CRC (Class-Responsibility-Collaborator) cards → UML diagrams → Pseudocode → Coding
UML Diagram

- **Top compartment:**
  - Name of class, Bolded, Centered

- **CreditCard**
  - customer : String
  - **<<constructor>>** CreditCard (String name)
  - + getCustomer() : String
  - + makePayment(amount : double)

Janyl Jumadinova  
Data Abstraction Software Development. Review of Java Constructs.  
September 14–16, 2020  
15 / 17
UML Diagram

- **Top compartment:**
  - Name of class, Bolded, Centered

- **Middle compartment:**
  - Attributes (data members)

- **Bottom compartment:**
  - Behaviors (member methods), method name, followed by parentheses
  - Plus (+) sign indicates public member method
UML Diagram

- **Top compartment:**
  - Name of class, Bolded, Centered

- **Middle compartment:**
  - Attributes (data members)

- **Bottom compartment:**
  - Behaviors (member methods), method name, followed by parentheses
  - Plus (+) sign indicates public member method
UML Diagram

- **Top compartment:**
  - Name of class, Bolded, Centered

- **Middle compartment:**
  - Attributes (data members)

- **Bottom compartment:**
  - Behaviors (member methods), method name, followed by parentheses
  - Plus (+) sign indicates public member method

---

**CreditCard**

- customer : String

...  

**<<constructor>>** CreditCard (String name)
+ getCustomer() : String
+ makePayment(amount : double)

...
Testing and Debugging

Testing:
Verify the correctness of a program on a representative subset of inputs.
- Handcrafted test suites.
- Randomly generated inputs.

Debugging:
- Print statements.
- Debugger (jdb).
Testing and Debugging

Testing:
Verify the correctness of a program on a representative subset of inputs.
- Handcrafted test suites.
- Randomly generated inputs.

Debugging:
- print statements.
- debugger (jdb).
A system that reverses sentences in a text.

More details about the behavior of this program are provided in the description of Project P-1.26 in the textbook.