Principles of Software Construction: Objects, Design and Concurrency

Exceptions, scope, static data and methods, and Generics

Christian Kästner  Charlie Garrod

© 2012-13  C Garrod, J Aldrich, and W Scherlis
Administrivia

- Homework 0 graded...
- Homework 1 due tonight
  - But you may use late days...
- Homework 2 coming soon
  - A virtual world
  - Due 14 February <3
Key concepts from Tuesday
Key concepts from Tuesday

• Method dispatch
• The java.lang.Object
• Exceptions
  - try/catch/finally
Key concepts for today

• Exceptions (continued)

• Java potpourri
  ▪ The `static` keyword
  ▪ Inner classes
  ▪ Scope and variable shadowing
  ▪ Generics
Recall the example:

```java
try {
    FileInputStream fileInput = new FileInputStream(filename);
    DataInput dataInput = new DataInputStream(fileInput);
    int i = dataInput.readInt();
    fileInput.close();
    return i;
} catch (FileNotFoundException e) {
    System.out.println("Could not open file " + filename);
    return -1;
} catch (IOException e) {
    System.out.println("Error reading binary data from file " + filename);
    return -1;
}
```
The finally keyword

- The finally block always runs after try/catch:

```java
try {
    System.out.println("Top");
    int[] a = new int[10];
    a[42] = 42;
    System.out.println("Bottom");
} catch (IndexOutOfBoundsException e) {
    System.out.println("Caught index out of bounds");
} finally {
    System.out.println("Finally got here");
}
```

- Prints:
  Top
  Caught index out of bounds
  Finally got here
The **finally** keyword, part 2

• The finally block always runs after try/catch:

```java
try {
    System.out.println("Top");
    int[] a = new int[10];
    a[2] = 2;
    System.out.println("Bottom");
} catch (IndexOutOfBoundsException e) {
    System.out.println("Caught index out of bounds");
} finally {
    System.out.println("Finally got here");
}
```

• Prints:
  Top
  Bottom
  Finally got here
Checked and unchecked exceptions

• Unchecked exception: any subclass of RuntimeException
  ▪ Indicates an error which is highly unlikely and/or typically unrecoverable

• Checked exception: any subclass of Exception but not RuntimeException
  ▪ Indicates an error that every caller should be aware of and explicitly decide to handle or pass on
Creating and throwing your own exceptions

- Methods must declare any checked exceptions they might throw

- If your class extends `java.lang.Exception` you can throw it:

```java
if (someErrorBlahBlahBlahBlah) {
    throw new MyCustomException("Blah blah blah");
}
```

- See `ReadFromFile` examples and `IllegalBowlingScoreException` and `ReadBowlingScore` example
Benefits of exceptions
Benefits of exceptions

- Provide high-level summary of error and stack trace
  - Compare: core dumped in C

- Can’t forget to handle common failure modes
  - Compare: using a flag or special return value

- Can optionally recover from failure
  - Compare: calling `System.exit()`

- Improve code structure
  - Separate routine operations from error-handling

- Allow consistent clean-up in both normal and exceptional operation
Guidelines for using exceptions

• Catch and handle all checked exceptions
  ▪ Unless there is no good way to do so...

• Use runtime exceptions for programming errors

• Other good practices
  ▪ Do not catch an exception without (at least somewhat) handling the error
  ▪ When you throw an exception, describe the error
  ▪ If you re-throw an exception, always include the original exception as the cause
Key concepts for today

- Exceptions (continued)
- Java potpourri
  - The static keyword
  - Inner classes
  - Scope and variable shadowing
  - Generics
Static data and methods

- A property is *static* if it is associated with the class (as opposed to being associated with an instance of the class)

- Examples
  - A simple Counter example
  - The main method – why is this static?
  - The java.lang.String valueOf methods
Inner classes in Java

- Classes can be defined inside other classes, or even inside class methods
  - e.g., A LinkedList.Node class accessible only from within a LinkedList class:

```java
public class LinkedList {
    private static class Node {
        public int val;
        public Node next;
        public Node(int v, Node n) {
            val = v;
            next = n;
        }
    }
    Node head;
    // ...
}
```
Variable shadowing in Java

- Variable shadowing: when a name within some program inner scope matches a name in some outer scope
  - e.g.,
  ```java
  public class Dog {
      String name;
      public Dog (String name) {
          this.name = name;
      }
  }
  ```

- Java has class variables, instance variables, and local variables
  - Local variables may shadow instance or class variables
  - A subclass’s variables may shadow superclass variables
Inner scopes in Java

- Curly braces define a new scope
  - e.g., public void printAsInt(String s) {
    try {
      Integer x = Integer.valueOf(s);
    } catch (NumberFormatException e) {
      // we ignore the exception
    }
    System.out.println(x); // x is undefined
  } // in this scope
Recall the Java Collection API (excerpt)
Consider the `java.util.Stack`

```java
public class Stack {
    public void push(Object obj) { ... }
    public Object pop() { ... }
}
```

- **Some possible client code?:**
  ```java
  Stack stack = new Stack();
  String s = "Hello!";
  stack.push(s);
  String t = stack.pop();
  ```
Consider the java.util.Stack

```java
public class Stack {
    public void push(Object obj) { ... }
    public Object pop() { ... }
}
```

- Some possible client code:
  ```java
  Stack stack = new Stack();
  String s = "Hello!";
  stack.push(s);
  String t = (String) stack.pop();
  ```

To fix the type error
Type polymorphism via Java Generics

• The java.util.Stack instead
  ▪ A stack of some type $T$:
    
    ```java
    public class Stack<T> {
        public void push(T obj) { … }
        public T pop() { … }
    }
    ```

  • Improves typechecking, simplifies(?) client code:
    
    ```java
    Stack<String> stack = new Stack<String>();
    String s = “Hello!”;
    stack.push(s);
    String t = stack.pop();
    ```