CSE 70: SE Principles and Java
Packages, Interfaces & Exceptions

Ingolf Krueger
Learning Goals for Today
Learning Goals

• Recap basics of OO and Java programming

• Understand the basics of encapsulation and information hiding

• Be able to utilize exceptions as a failure management technique

• Be able to write Java code that has
  – packages,
  – interfaces and
  – exceptions.
Basics of Object Orientation
Classes and Objects

- Classes are one of the key means for structuring and abstraction in OO

- A class models a concept of the application domain / implementation

- Each object is an instance of a class synonyms: object, instance, class instance, ...

- Classes define templates for individual objects
Basics of Java
Basic Structure of a Java Program

// package declarations
package chatsystem;

// import statements
import java.util.LinkedList;
...

public class ChatServer {
  // fields
  private LinkedList<String> clients;
  ...
  // methods
  public void sign_on(String client_id) {
    clients.add(client_id);
  }
  ...
  public static void main(String[] args) {
    ...
  }
}
Java Elements

• Package Declaration
  – Projects consist of multiple source files
  – Package declarations structure these files into groups of related files
  – More about this later

• Import Statements
  – Identify which elements from another file are going to be visible in this file

• Class Declaration
  – Identifies Data belonging to a class
  – Identifies Operations (Methods) available to manipulate the data

• Main
  – Special method – serves as the (single) entry point into an application
How to Design and Start a Computation

• Write Classes
  – Each class identifies a concept in your system (Examples: ChatServer, ChatClient, AddressBook, BuddyList)
  – Identify the data each object of the class (instance) is going to manipulate
  – Offer functions to manipulate the data
  – Provide constructors, i.e. special “methods” called to initialize a new instance of the class

• One Class will contain method `main`
  – Use it to create objects and execute their methods
  – You create an object by calling `new` with a class name, and constructor arguments
  – This sets the computation in motion
Example: Starting a Computation

```java
package chatsystem;

public class ChatSystem {

    private ChatSystem system;
    private ChatServer server;
    private ChatClient client;
    ...

    public static void main(String[] args) {
        system = new ChatSystem(“CSE 70”);
        server = new ChatServer(“servername”);
        client = new ChatClient(“username”);
        system.add(server);
        system.add(client);
        server.connect(client);
    }

```
How to Compute

• Assignments
  – Change the value/state of a variable/field

• Control Structures
  – if: conditional execution
  – while: loop while condition holds
  – for: loop for specified range, with initialization
  – others:
    • case/switch
    • try/catch/finally
    • ...
Example: If & For-Loop

package chatsystem;

public class ChatSystem {

    private ChatClient[10] clients;
    ...
    public static void main(String[] args) {
        ...
        ...
        for(int i = 0; i < 10; i++) {
            if(clients[i] != null)
                clients[i].handleMessage(“Happy New Year!”);
        }
        ...
    }
}
Your Turn!
Work with your Neighbors for 7 Minutes

• Introduce yourself to your neighbors

• On a piece of paper, concisely write down your top question on, or problem with, Java/OO

• If any of your neighbors knows the answer, have them write it down in two sentences max

• Pass the paper to the isle, we’ll read a few now, and take the others for the discussion section on Monday!
Important Software Engineering Principles: Encapsulation/Information Hiding
One Information Hiding Mechanism in Java: Interfaces

• Say only **what** you plan to implement, not **how**

• Abstract away from implementation details/give implementation later

• Can change the implementation while keeping the same interface

• **Java interface:**
  – Collection of abstract methods – no method bodies allowed
  – Acts as a specification/contract between a client and an implementation of the interface

• Classes can implement any number of interfaces

• Class implementing an interface must provide body for all methods defined in the interface
public interface ChatServer {

    public void sign_on(String client_id);

    public void publish(String message);

    public boolean isPresent(String client_id);

}
Example: ChatServer Interface, First Implementation

```java
public class ChatServerImpl1 implements ChatServer {

  ...  

  private String[] clients;
  private int client_index;
  ...  

  public void sign_on(String client_id) {
    clients[client_index++] = client_id;
  }

  ...  
}
```
Example: ChatServer Interface, Second Implementation

```java
public class ChatServerImpl2 implements ChatServer {
    ...
    private LinkedList<String> clients;
    ...
    public void sign_on(String client_id) {
        clients.add(client_id);
    }
    ...
}
```

ChatServerImpl2.java
Important Software Engineering Practice:
Structure your code so you (and others) can find their way around in it.
One Structuring Mechanism in Java: Packages

• Structure your classes and interfaces into coherent groups.

• Limit visibility of your classes and interfaces (package/friendly visibility).

• Define namespaces for your classes and interfaces

• Organize your .java and .class files according to package structure.
Example: Chat System

• We’ll develop

  – Classes and interfaces belonging to the chat client

  – Classes and interfaces belonging to the chat server

  – Possibly classes and interfaces belonging to both

• Keeping all of these in the same source directory will quickly overwhelm our ability to stay on top of what is (defined/stored) where.
Idea: Organize the Chat System into Packages

- Candidate packages:
  - chat.server
  - chat.client
  - chat.common
- Package names are “.”-separated lists of names

```java
package chat.server;

public interface ChatServer {
    public void sign_on(String client_id);
    public void publish(String message);
    public boolean isPresent(String client_id);
}
```

ChatServer.java

© Ingolf Krueger, 2010
Idea: Organize the Chat System into Packages

- Candidate packages:
  - chat.server
  - chat.client
  - chat.common

- Package names are “.”-separated lists of names

package chat.server;

public class ChatServerImpl implements ChatServer {
  ...
  public void sign_on(String client_id) {
    clients.put(client_id, client);
  }
  ...
}
Idea: Organize the Chat System into Packages

- Candidate packages:
  - chat.server
  - chat.client
  - chat.common

- Package names are “.”-separated lists of names

```java
package chat.client;

public interface ChatClient {
    public void notify(String m);
}
```

ChatClient.java
Reflect Java Packages in the Directory Structure

```java
package chat.server;

public interface ChatServer
{
    ... 
}

package chat.server;

public class ChatServerImpl
{
    ... 
}

package chat.client;

public interface ChatClient
{
    ... 
}
```

<your source path> /src

- chat
  - server
    - ChatServer.java
    - ChatServerImpl.java
  - client
    - ChatClient.java
    - ChatClientImpl.java
Reflect Java Packages in the Directory Structure

```java
package chat.server;

public interface ChatServer {
    ...
}

package chat.server;

public class ChatServerImpl {
    ...
}

package chat.client;

public interface ChatClient {
    ...
}
```
How to Access Elements of a Package

package chat.server;

public interface ChatServer
{
   ...
}

package chat.client;

public interface ChatClient
{
   ...
}

package chat.system;

import chat.server.ChatServer;  // selective import
import chat.client.*;          // everything

ChatSystem.java
Example: Overall Directory Structure

<your project path>

```
src
  chat
  bin
  chat
  ...  client
  ...  server
  ...  client
  ...  server
```

© Ingolf Krueger, 2010
Important Software Engineering Practice: Use *Exceptions* instead of error codes
Your Turn!
Team up with your neighbors for 3 minutes

- What can go wrong here?
- How do we mark “dangerous” code?
- How do we communicate to client code that it happened?

```java
public interface IntDivider {
    public int divide(int x, int y);
}
```

```java
public class IntDividerImpl implements IntDivider {
    public int divide(int x, int y) {
        return x/y;
    }
}
```
Solution: Introduce Java Exception

```java
public class DivideByZeroAttempt extends Exception {

    public DivideByZeroAttempt() {
        super();
    }

    public DivideByZeroAttempt(String message) {
        super(message);
    }
}
```
public interface IntDivider {
    public int divide(int x, int y) throws DivideByZeroAttempt;
}

public class IntDividerImpl implements IntDivider {
    public int divide(int x, int y) throws DivideByZeroAttempt{
        if (y == 0)
            throw new DivideByZeroAttempt("tragic");
        return x/y;
    }
}
Client Code is Forewarned and Can Handle Exception

```java
... public static void main(String[] args) {
    IntDivider id = new IntDividerImpl();
    try {
        id.divide(3, 2);
        id.divide(3, 0);
        id.divide(2, 3);
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

DivideByZeroAttempt.java
What have you learned today?
Learning Goals

• Recap basics of OO and Java programming

• Understand the basics of encapsulation and information hiding

• Be able to utilize exceptions as a failure management technique

• Be able to write Java code that has
  – packages,
  – interfaces and
  – exceptions.