Controlled interaction with objects

Madhavan Mukund

`https://www.cmi.ac.in/~madhavan`

Programming Concepts using Java

Week 4

Manipulating objects

- Encapsulation is a key principle of object oriented programming
  - Internal data is private
  - Access to the data is regulated through public methods
  - Accessor and mutator methods

```
public class Date {
  private int day, month year;

  public void getDay(int d) {...}
  public void getMonth(int m) {...}
  public void getYear(int y) {...}

  public void setDay(int d) {...}
  public void setMonth(int m) {...}
  public void setYear(int y) {...}
}
```

Manipulating objects

- Encapsulation is a key principle of object oriented programming
  - Internal data is private
  - Access to the data is regulated through public methods
  - Accessor and mutator methods
- Can ensure data integrity by regulating access

```
public class Date {
  private int day, month year;

  public void getDay(int d) {...}
  public void getMonth(int m) {...}
  public void getYear(int y) {...}

  public void setDay(int d) {...}
  public void setMonth(int m) {...}
  public void setYear(int y) {...}
}
```

Manipulating objects

- Encapsulation is a key principle of object oriented programming
  - Internal data is private
  - Access to the data is regulated through public methods
  - Accessor and mutator methods
- Can ensure data integrity by regulating access
- Update date as a whole, rather than individual components

```
public class Date {
  private int day, month year;

  public void getDay(int d) {...}
  public void getMonth(int m) {...}
  public void getYear(int y) {...}

  public void setDate(int d, int m, int y) {
    ...
    // Validate d-m-y combination
  }

}
```

Manipulating objects

- Encapsulation is a key principle of object oriented programming
  - Internal data is private
  - Access to the data is regulated through public methods
  - Accessor and mutator methods
- Can ensure data integrity by regulating access
- Update date as a whole, rather than individual components
- Does this provide sufficient control?

```
public class Date {
  private int day, month year;

  public void getDay(int d) {...}
  public void getMonth(int m) {...}
  public void getYear(int y) {...}

  public void setDate(int d, int m, int y) {
    ...
    // Validate d-m-y combination
  }

}
```

Querying a database

- Object stores train reservation information
  - Can query availability for a given train, date

```
public class RailwayBooking {
  private BookingDB railwaydb;

  public int getStatus(int trainno, Date d) {
    // Return number of seats available
    // on train number trainno on date d
    ...
  }
}
```

Querying a database

- Object stores train reservation information
    - Can query availability for a given train, date
- To control spamming by bots, require user to log in before querying

```
public class RailwayBooking {
  private BookingDB railwaydb;

  public int getStatus(int trainno, Date d) {
    // Return number of seats available
    // on train number trainno on date d
    ...
  }
}
```

Querying a database

- Object stores train reservation information
  - Can query availability for a given train, date
- To control spamming by bots, require user to log in before querying
- Need to connect the query to the logged in status of the user

```
public class RailwayBooking {
  private BookingDB railwaydb;

  public int getStatus(int trainno, Date d) {
    // Return number of seats available
    // on train number trainno on date d
    ...
  }
}
```

Querying a database

- Object stores train reservation information
  - Can query availability for a given train, date
- To control spamming by bots, require user to log in before querying
- Need to connect the query to the logged in status of the user
- "Interaction with state"

```
public class RailwayBooking {
  private BookingDB railwaydb;

  public int getStatus(int trainno, Date d) {
    // Return number of seats available
    // on train number trainno on date d
    ...
  }
}
```

Querying a database

- Need to connect the query to the logged in status of the user

```
public class RailwayBooking {
  private BookingDB railwaydb;

  public int getStatus(int trainno, Date d) {
    // Return number of seats available
    // on train number trainno on date d
    ...
  }
}
```

Querying a database

- Need to connect the query to the logged in status of the user
- Use objects!
  - On log in, user receives an object that can make a query
  - Object is created from private class that can look up `railwaydb`

```
public class RailwayBooking {
  private BookingDB railwaydb;

  public QueryObject login(String u, String p){
    QueryObject qobj;
    if (valid_login(u,p)) {
       qobj = new QueryObject();
       return(qobj);
    }
  }

  private class QueryObject {
    public int getStatus(int trainno, Date d) {
      // Return number of seats available
      // on train number trainno on date d
      ...
    }
  }
}
```

Querying a database

- Need to connect the query to the logged in status of the user
- Use objects!
    - On log in, user receives an object that can make a query
    - Object is created from private class that can look up `railwaydb`
- How does user know the capabilities of private class `QueryObject`?

```
public class RailwayBooking {
  private BookingDB railwaydb;

  public QueryObject login(String u, String p){
    QueryObject qobj;
    if (valid_login(u,p)) {
       qobj = new QueryObject();
       return(qobj);
    }
  }

  private class QueryObject {
    public int getStatus(int trainno, Date d) {
      // Return number of seats available
      // on train number trainno on date d
      ...
    }
  }
}
```

Querying a database

- Need to connect the query to the logged in status of the user
- Use objects!
  - On log in, user receives an object that can make a query
  - Object is created from private class that can look up `railwaydb`
- How does user know the capabilities of private class `QueryObject`?
- Use an interface!
  - Interface describes the capability of the object returned on login

```
public interface QIF{
  public abstract int
    getStatus(int trainno, Date d);
}

public class RailwayBooking {
  private BookingDB railwaydb;
  public QIF login(String u, String p){
    QueryObject qobj;
    if (valid_login(u,p)) {
       qobj = new QueryObject();
       return(qobj);
    }
  }
  private class QueryObject implements QIF {
    public int getStatus(int trainno, Date d){
      ...
    }
  }
}
```

Querying a database

- Query object allows unlimited number of queries

```
public interface QIF{
  public abstract int
    getStatus(int trainno, Date d);
}

public class RailwayBooking {
  private BookingDB railwaydb;
  public QIF login(String u, String p){
    QueryObject qobj;
    if (valid_login(u,p)) {
       qobj = new QueryObject();
       return(qobj);
    }
  }
  private class QueryObject implements QIF {
    public int getStatus(int trainno, Date d){
     ...
    }
  }
}
```

Querying a database

- Query object allows unlimited number of queries
- Limit the number of queries per login?

```
public interface QIF{
  public abstract int
    getStatus(int trainno, Date d);
}

public class RailwayBooking {
  private BookingDB railwaydb;
  public QIF login(String u, String p){
    QueryObject qobj;
    if (valid_login(u,p)) {
       qobj = new QueryObject();
       return(qobj);
    }
  }
  private class QueryObject implements QIF {
    public int getStatus(int trainno, Date d){
     ...
    }
  }
}
```

Querying a database

- Query object allows unlimited number of queries
- Limit the number of queries per login?
- Maintain a counter
  - Add instance variables to object returned on login
  - Query object can remember the state of the interaction

```
public class RailwayBooking {
  private BookingDB railwaydb;
  public QIF login(String u, String p){
    QueryObject qobj;
    if (valid_login(u,p)) {
       qobj = new QueryObject();
       return(qobj);
    }
  }
  private class QueryObject implements QIF {
    private int numqueries;
    private static int QLIM;

    public int getStatus(int trainno, Date d){
      if (numqueries < QLIM){
        // respond, increment numqueries
      }
    }
  }
}
```

Summary

- Can provide controlled access to an object
- Combine private classes with interfaces
- External interaction is through an object of the private class
- Capabilities of this object are known through a public interface
- Object can maintain instance variables to track the state of the interaction