



CHAPTER 1

Removing Duplicate Code



How duplicate code may be created

Consider the following code:

```
public class BookRental {
    String id;
    String customerName;
    ...
}
public class BookRentals {
    private Vector rentals;
    public String getCustomerName(String rentalId) {
        for (int i = 0; i < rentals.size(); i++) {
            BookRental rental = (BookRental) rentals.elementAt(i);
            if (rental.getId().equals(rentalId)) {
                return rental.getCustomerName();
            }
        }
        throw new RentalNotFoundException();
    }
}
public class RentalNotFoundException extends Exception {
    ...
}
```

Suppose that you need to add a new method to delete a rental given its id. You figure that you need to search for the rental one by one, just like what you did in `getCustomerName`. So you copy and paste `getCustomerName` and modify the result to get the new method:

```
public class BookRentals {
    private Vector rentals;
    public String getCustomerName(String rentalId) {
        for (int i = 0; i < rentals.size(); i++) {
            BookRental rental = (BookRental) rentals.elementAt(i);
            if (rental.getId().equals(rentalId)) {
                return rental.getCustomerName();
            }
        }
        throw new RentalNotFoundException();
    }
    public void deleteRental(String rentalId) {
        for (int i = 0; i < rentals.size(); i++) {
            BookRental rental = (BookRental) rentals.elementAt(i);
            if (rental.getId().equals(rentalId)) {
                rentals.remove(i);
                return;
            }
        }
    }
}
```

```

        }
    }
    throw new RentalNotFoundException();
}
}

```

Does the code look fine? No, the two methods share quite a lot of code (code duplication).

Removing duplicate code

To remove the duplication, you can change the BookRentals class ("refactor" it):

```

public class BookRentals {
    private Vector rentals;
    public String getCustomerName(String rentalId) {
        int rentalIdx = getRentalIdxById(rentalId);
        return ((BookRental) rentals.elementAt(rentalIdx)).getCustomerName();
    }
    public void deleteRental(String rentalId) {
        rentals.remove(getRentalIdxById(rentalId));
    }
    private int getRentalIdxById(String rentalId) {
        for (int i = 0; i < rentals.size(); i++) {
            BookRental rental = (BookRental) rentals.elementAt(i);
            if (rental.getId().equals(rentalId)) {
                return i;
            }
        }
        throw new RentalNotFoundException();
    }
}
}

```

Why remove duplicate code

Why remove duplicate code? Suppose that you would like to change the "rentals" field from a vector into an array, you need to change "rentals.size()" to "rentals.length". In the refactored version you only need to change it once in getRentalIdxById, while in the original version you need to change it twice in getCustomerName and deleteRental. Similarly, you need to change "rentals.elementAt(i)" to "rentals[i]". In the refactored version you do it only once, while in the original version you do it twice.

In general, if the same code is duplicated in 10 places, when you need to change the code, you will have to find all these 10 places and change them all. If you forget to change some places, in this particular case (change from vector to array) the compiler will detect the errors, but in some other cases they will become bugs that will take a lot of your precious time to find.

Licensed for viewing only. Printing is prohibited. For hard copies, please purchase from www.agilekills.org

References

- <http://c2.com/cgi/wiki?OnceAndOnlyOnce>.
- <http://www.martinfowler.com/ieeeSoftware/repetition.pdf>.
- A. Hunt, and D. Thomas, Pragmatic Programmer, Addison Wesley, 1999.
- Kent Beck, Extreme Programming Explained, Addison-Wesley, 2000.



Chapter exercises

Introduction

1. The solutions should be in the format of source code in Java, C++ or Delphi. It does not have to be free of syntax errors as long as it can communicate the design clearly. If required, add some text explanations to justify the design.
2. For some problems, there are hints at the end of the problems. However, you should try to solve the problems yourself first! Look at the hints only if you are desperate.
3. After the hints, there are sample solutions. After working out your own solutions, you should compare them to yours.

Problems

1. Point out and remove the duplication in the code:

```
class Organization {
    String id;
    String eName; //English name
    String cName; //Chinese name
    String telCountryCode;
    String telAreaCode;
    String telLocalNumber;
    String faxCountryCode;
    String faxAreaCode;
    String faxLocalNumber;
    String contactPersonEFirstName; //First name and last name in English
    String contactPersonELastName;
    String contactPersonCFirstName; //First name and last name in Chinese
    String contactPersonCLastName;
    String contactPersonTelCountryCode;
    String contactPersonTelAreaCode;
    String contactPersonTelNumber;
    String contactPersonFaxCountryCode;
    String contactPersonFaxAreaCode;
    String contactPersonFaxLocalNumber;
    String contactPersonMobileCountryCode;
    String contactPersonMobileAreaCode;
    String contactPersonMobileLocalNumber;
    ...
}
```

2. Point out and remove the duplication in the code:

```
class ParticipantsInDB {
    Connection db;
    ParticipantsInDB(){
```

```

    Class.forName("org.postgresql.Driver");
    db =
        DriverManager.getConnection(
            "jdbc:postgresql://myhost/ConferenceDB",
            "PaulChan",
            "myP@ssword");
    }
    void addParticipant(Participant part) {
        PreparedStatement st = db.prepareStatement("INSERT INTO participants
VALUES (?, ?, ?, ?, ?, ?, ?)");
        try {
            st.setString(1, part.getId());
            st.setString(2, part.getEFirstName());
            st.setString(3, part.getELastName());
            ...
            st.executeUpdate();
        } finally {
            st.close();
        }
    }
}
class OrganizationsInDB {
    Connection db;
    OrganizationsInDB() {
        Class.forName("org.postgresql.Driver");
        db =
            DriverManager.getConnection(
                "jdbc:postgresql://myhost/ConferenceDB",
                "PaulChan",
                "myP@ssword");
    }
    void addOrganization(Organization o) {
        PreparedStatement st =
            db.prepareStatement(
                "INSERT INTO organizations VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)");
        try {
            st.setString(1, o.getId());
            st.setString(2, o.getEName());
            st.setString(3, o.getCName());
            ...
            st.executeUpdate();
        } finally {
            st.close();
        }
    }
}
}

```

3. Point out and remove the duplication in the code:

```

class ParticipantsInDB {
    Connection db;
    void addParticipant(Participant part) {
        PreparedStatement st = db.prepareStatement("INSERT INTO participants
VALUES (?, ?, ?, ?, ?, ?, ?)");
        try {
            st.setString(1, part.getId());
            ...
            st.executeUpdate();
        } finally {

```

```

        st.close();
    }
}
void deleteAllParticipants() {
    PreparedStatement st = db.prepareStatement("DELETE FROM participants");
    try {
        st.executeUpdate();
    } finally {
        st.close();
    }
}
int getCount() {
    PreparedStatement st = db.prepareStatement("SELECT COUNT(*) FROM
participants");
    try {
        ResultSet rs = st.executeQuery();
        rs.next();
        return rs.getInt(1);
    } finally {
        st.close();
    }
}
}

```

4. Point out and remove the duplication in the code:

```

class ParticipantsInDBTest extends TestCase {
    ParticipantsInDB p;
    void setUp() {
        p=ParticipantsInDB.getInstance();
    }
    void tearDown() {
        ParticipantsInDB.freeInstance();
    }
    void testAdd() {
        Participant part1=new Participant("ABC001","Kent","Tong",true,"Manager");
        p.deleteAllParticipants();
        p.addParticipant(part1);
        assertEquals(p.getCount(),1);
    }
    void testAdd2() {
        Participant part1=new Participant("ABC001","Kent","Tong",true,"Manager");
        Participant part2=new Participant("ABC003","Paul","Chan",true,"Manager");
        p.deleteAllParticipants();
        p.addParticipant(part1);
        p.addParticipant(part2);
        assertEquals(p.getCount(),2);
    }
    void testEnum() {
        Participant part1=new Participant("ABC001","Kent","Tong",true,"Manager");
        Participant part2=new Participant("ABC003","Paul","Chan",true,"Manager");
        p.deleteAllParticipants();
        p.addParticipant(part2);
        p.addParticipant(part1);
        ParticipantEnumeratorById penum=new ParticipantEnumeratorById();
        assertTrue(penum.next());
        assertTrue(penum.get().equals(part1));
        assertTrue(penum.next());
        assertTrue(penum.get().equals(part2));
    }
}

```

```
    assertTrue(!penum.next());
    penum.close();
}
}
```

5. Point out and remove the duplication in the code:

```
class ReportCatalogueIndexCommandParser {
    final String NO_GROUPING = "orgNoGrouping";
    static final int ORG_CATALOG = 0;
    static final int PART_CATALOG = 1;
    int getGroupingType(String grouping) {
        if (grouping.equals(NO_GROUPING)) {
            return ORG_CATALOG;
        } else if (grouping.equals("orgGroupByCountry")) {
            return ORG_CATALOG;
        } else if (grouping.equals("orgGroupByTypeOfOrgName")) {
            return ORG_CATALOG;
        } else if (grouping.equals("part")) {
            return PART_CATALOG;
        } else
            throw new IllegalArgumentException("Invalid grouping!");
    }
    ...
}
```

6. Point out and remove the duplication in the code:

```
class Restaurant extends Account {
    //the string "Rest" is concated with the restaurant ID to
    //form the key.
    final static String RestaurantIDText = "Rest";
    String website;
    //address in Chinese.
    String addr_cn;
    //address in English.
    String addr_en;
    //the restaurant would like to update its fax # with this. After it is
    //confirmed, it will be stored in Account. Before that, it is stored
    //here.
    String numb_newfax;
    boolean has_NewFax = false;
    //a list of holidays.
    Vector Holiday; // a holiday
    //id of the category this restaurant belongs to.
    String catId;
    //a list of business session. Each business session is an array
    //of two times. The first time is the start time. The second time
    //is the end time. The restaurant is open for business in each
    //session.
    Vector BsHour; //Business hour
    ...
    //y: year.
    //m: month.
    //d: date.
    void addHoliday(int y,int m,int d){
        if(y<1900) y+=1900;
        Calendar aHoliday = (new GregorianCalendar(y,m,d,0,0,0));
    }
}
```

```

        Holiday.add(aHoliday);
    }
    public boolean addBsHour(int fromHr, int fromMin, int toHr, int toMin){
        int fMin = fromHr*60 + fromMin; //start time in minutes.
        int tMin = toHr*60 + toMin; //end time in minutes.
        //make sure both times are valid and the start time is earlier
        //than the end time.
        if(fMin > 0 && fMin <= 1440 && tMin > 0 && tMin <=1440 && fMin < tMin){
            GregorianCalendar bs[] = {
                new GregorianCalendar(1900,1,1, fromHr, fromMin,0),
                new GregorianCalendar(1900,1,1, toHr, toMin,0)
            };
            BsHour.add(bs);
            return true;
        } else {
            return false;
        }
    }
}

```

7. Point out and remove the duplication in the code:

```

class Order {
    ...
    boolean IsSameString(String s1, String s2){
        if(s1==s2) return true;
        if(s1==null) return false;
        return(s1.equals(s2));
    }
}
class Mail {
    ...
    static boolean IsSameString(String s1, String s2) {
        if (s1 == s2)
            return true;
        if (s1 == null)
            return false;
        return (s1.equals(s2));
    }
}

```

8. Point out and remove the duplication in the code:

```

class FoodDB {
    //find all foods whose names contain both the keywords. returns
    //an iterator on these foods.
    public Iterator srchFood(String cName, String eName){
        //it contains all the foods to be returned.
        TreeMap foodTree = new TreeMap();
        Iterator foodList;
        Food food;
        foodList = getAllRecords();
        while (foodList.hasNext()){
            food = (Food) foodList.next();
            //do its names contain both keywords?
            if ((cName==null || //null or "" means no key is specified
                cName.equals("") ||
                food.getCName().indexOf(cName)!=-1)

```

Licensed for viewing only. Printing is prohibited. For hard copies, please purchase from www.agileskills.org

```

        &&
        (eName==null || //null or "" means no key is specified
         eName.equals("") ||
         food.getEName().indexOf(eName)!=-1){
            foodTree.put(food.getFoodKey(), food);
        }
    }
    return foodTree.values().iterator();
}
}

```

9. Point out and remove the duplication in the code:

```

class UIDialogCustomerMain extends JDialog {
    JButton btnOrderDel;
    JButton btnCustChangePassword;
    void bindEvents() {
        ...
        btnOrderDel.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Dialog d = new UIDialogCustomerDeleteOrder(UIDialogCustomerMain.this,
"Del Order", true);
                d.pack();
                d.setLocation(400,200);
                d.setVisible(true);
            }
        });
        btnCustChangePassword.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Dialog d = new UIChangeAccountPW(UIDialogCustomerMain.this, "chg pw",
true);
                d.pack();
                d.setLocation(400,200);
                d.setVisible(true);
            }
        });
    }
    ...
}

class UIDialogRestaurantMain extends JDialog {
    JButton btnChangePassword;
    void bindEvents() {
        ...
        btnChangePassword.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Dialog d = new UIChangeAccountPW(UIDialogRestaurantMain.this, "chg
pw", true);
                d.pack();
                d.setLocation(400,200);
                d.setVisible(true);
            }
        });
    }
    ...
}

```

10. Suppose that there are two kinds of rentals: book and movie. Point out and remove the duplication in the code:

```
public class BookRental {
    private String bookTitle;
    private String author;
    private Date rentDate;
    private Date dueDate;
    private double rentalFee;
    public boolean isOverdue() {
        Date now=new Date();
        return dueDate.before(now);
    }
    public double getTotalFee() {
        return isOverdue() ? 1.2*rentalFee : rentalFee;
    }
}
public class MovieRental {
    private String movieTitle;
    private int classification;
    private Date rentDate;
    private Date dueDate;
    private double rentalFee;
    public boolean isOverdue() {
        Date now=new Date();
        return dueDate.before(now);
    }
    public double getTotalFee() {
        return isOverdue() ? 1.3*rentalFee : rentalFee;
    }
}
```

11. Come up with some code that contains duplicate code/structure, point out the duplication and eliminate it.

Hints

1. The triple (country code, area code, local number) appears several times. It should be extracted into a class such as "TelNo". The word "contactPerson" is duplicated in many fields. The fields should be extracted into a class such as "ContactPerson".
2. The code to load the PostgreSQL JDBC driver and obtain a connection is duplicated in both classes. The code should be extracted into a class such as "ConferenceDBConnection".
3. The table name (i.e., "participants") is duplicated in the three methods. It should be extracted into a static final String in the class or into a class such as "ParticipantsTable".
4. The participant objects are duplicate. They should be extracted to become instance variables.
5. Each branch of if-then is doing similar thing. You may extract the strings used in the conditions into an array or a map. Then just do a lookup instead of multiple if-then's.

Sample solutions

1. Point out and remove the duplication in the code:

```
class Organization {
    String id;
    String eName; //English name
    String cName; //Chinese name
    String telCountryCode;
    String telAreaCode;
    String telLocalNumber;
    String faxCountryCode;
    String faxAreaCode;
    String faxLocalNumber;
    String contactPersonEFirstName; //First name and last name in English
    String contactPersonELastName;
    String contactPersonCFirstName; //First name and last name in Chinese
    String contactPersonCLastName;
    String contactPersonTelCountryCode;
    String contactPersonTelAreaCode;
    String contactPersonTelNumber;
    String contactPersonFaxCountryCode;
    String contactPersonFaxAreaCode;
    String contactPersonFaxLocalNumber;
    String contactPersonMobileCountryCode;
    String contactPersonMobileAreaCode;
    String contactPersonMobileLocalNumber;
    ...
}
```

Turn the code into:

```
class Organization {
    String id;
    String eName;
    String cName;
    TelNo telNo;
    TelNo faxNo;
    ContactPerson contactPerson;
    ...
}

class ContactPerson{
    String eFirstName;
    String eLastName;
    String cFirstName;
    String cLastName;
    TelNo tel;
    TelNo fax;
    TelNo mobile;
}

class TelNo {
    String countryCode;
    String areaCode;
    String localNumber;
}
```

If you are worried that the pair (FirstName, LastName) appears twice, you may further extract the pair into a class:

```
class ContactPerson{
    FullName eFullName;
    FullName cFullName;
    TelNo tel;
    TelNo fax;
    TelNo mobile;
}
class FullName {
    String firstName;
    String lastName;
}
```

2. Point out and remove the duplication in the code:

```
class ParticipantsInDB {
    Connection db;
    ParticipantsInDB() {
        Class.forName("org.postgresql.Driver");
        db =
            DriverManager.getConnection(
                "jdbc:postgresql://myhost/ConferenceDB",
                "PaulChan",
                "myP@ssword");
    }
    void addParticipant(Participant part) {
        PreparedStatement st = db.prepareStatement("INSERT INTO participants VALUES
(?, ?, ?, ?, ?, ?, ?)");
        try {
            st.setString(1, part.getId());
            st.setString(2, part.getEFirstName());
            st.setString(3, part.getELastName());
            ...
            st.executeUpdate();
        } finally {
            st.close();
        }
    }
}
class OrganizationsInDB {
    Connection db;
    OrganizationsInDB() {
        Class.forName("org.postgresql.Driver");
        db =
            DriverManager.getConnection(
                "jdbc:postgresql://myhost/ConferenceDB",
                "PaulChan",
                "myP@ssword");
    }
    void addOrganization(Organization o) {
        PreparedStatement st =
            db.prepareStatement(
                "INSERT INTO organizations VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)");
        try {
            st.setString(1, o.getId());
```

```

        st.setString(2, o.getEName());
        st.setString(3, o.getCName());
        ...
        st.executeUpdate();
    } finally {
        st.close();
    }
}
}

```

The code to setup the DB connection is duplicated. Extract it into a separate class:

```

class ConferenceDBConnection {
    static Connection makeConnection() {
        Class.forName("org.postgresql.Driver");
        return
            DriverManager.getConnection(
                "jdbc:postgresql://myhost/ConferenceDB",
                "PaulChan",
                "myP@sssword");
    }
}
class ParticipantsInDB {
    Connection db;
    ParticipantsInDB(){
        db = ConferenceDBConnection.makeConnection();
    }
    void addParticipant(Participant part) {
        PreparedStatement st = db.prepareStatement("INSERT INTO participants
VALUES (?, ?, ?, ?, ?, ?, ?)");
        try {
            st.setString(1, part.getId());
            st.setString(2, part.getEFirstName());
            st.setString(3, part.getELastName());
            ...
            st.executeUpdate();
        } finally {
            st.close();
        }
    }
}
class OrganizationsInDB {
    Connection db;
    OrganizationsInDB() {
        db = ConferenceDBConnection.makeConnection();
    }
    void addOrganization(Organization o) {
        PreparedStatement st =
            db.prepareStatement(
                "INSERT INTO organizations VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)");
        try {
            st.setString(1, o.getId());
            st.setString(2, o.getEName());
            st.setString(3, o.getCName());
            ...
            st.executeUpdate();
        } finally {
            st.close();
        }
    }
}

```

```

}
}

```

3. Point out and remove the duplication in the code:

```

class ParticipantsInDB {
    Connection db;
    void addParticipant(Participant part) {
        PreparedStatement st = db.prepareStatement("INSERT INTO participants VALUES
(?, ?, ?, ?, ?, ?, ?)");
        try {
            st.setString(1, part.getId());
            ...
            st.executeUpdate();
        } finally {
            st.close();
        }
    }
    void deleteAllParticipants() {
        PreparedStatement st = db.prepareStatement("DELETE FROM participants");
        try {
            st.executeUpdate();
        } finally {
            st.close();
        }
    }
    int getCount() {
        PreparedStatement st = db.prepareStatement("SELECT COUNT(*) FROM
participants");
        try {
            ResultSet rs = st.executeQuery();
            rs.next();
            return rs.getInt(1);
        } finally {
            st.close();
        }
    }
}

```

The table name is duplicated. The call to "db.preparedStatement" is duplicated.

```

class ParticipantsInDB {
    static final String tableName="participants";
    Connection db;
    PreparedStatement makeStatement(String sql) {
        return db.prepareStatement(sql);
    }
    void addParticipant(Participant part) {
        PreparedStatement st = makeStatement("INSERT INTO "+tableName+" VALUES
(?, ?, ?, ?, ?, ?, ?)");
        try {
            st.setString(1, part.getId());
            ...
            st.executeUpdate();
        } finally {
            st.close();
        }
    }
}

```

```
void deleteAllParticipants() {
    PreparedStatement st = makeStatement("DELETE FROM "+tableName);
    try {
        st.executeUpdate();
    } finally {
        st.close();
    }
}

int getCount() {
    PreparedStatement st = makeStatement("SELECT COUNT(*) FROM "+tableName);
    try {
        ResultSet rs = st.executeQuery();
        rs.next();
        return rs.getInt(1);
    } finally {
        st.close();
    }
}
}
```

4. Point out and remove the duplication in the code:

```
class ParticipantsInDBTest extends TestCase {
    ParticipantsInDB p;
    void setUp() {
        p = ParticipantsInDB.getInstance();
    }
    void tearDown() {
        ParticipantsInDB.freeInstance();
    }
    void testAdd() {
        Participant part1 =
            new Participant("ABC001", "Kent", "Tong", true, "Manager");
        p.deleteAllParticipants();
        p.addParticipant(part1);
        assertEquals(p.getCount(), 1);
    }
    void testAdd2() {
        Participant part1 =
            new Participant("ABC001", "Kent", "Tong", true, "Manager");
        Participant part2 =
            new Participant("ABC003", "Paul", "Chan", true, "Manager");
        p.deleteAllParticipants();
        p.addParticipant(part1);
        p.addParticipant(part2);
        assertEquals(p.getCount(), 2);
    }
    void testEnum() {
        Participant part1 =
            new Participant("ABC001", "Kent", "Tong", true, "Manager");
        Participant part2 =
            new Participant("ABC003", "Paul", "Chan", true, "Manager");
        p.deleteAllParticipants();
        p.addParticipant(part2);
        p.addParticipant(part1);
        ParticipantEnumeratorById penum = new ParticipantEnumeratorById();
        assertTrue(penum.next());
        assertTrue(penum.get().equals(part1));
        assertTrue(penum.next());
    }
}
```

```

        assertTrue(penum.get().equals(part2));
        assertTrue(!penum.next());
        penum.close();
    }
}

```

The two participant objects are duplicate.

```

class ParticipantsInDBTest extends TestCase {
    ParticipantsInDB p;
    Participant part1=new Participant("ABC001", "Kent", "Tong", true, "Manager");
    Participant part2=new Participant("ABC003", "Paul", "Chan", true, "Manager");
    void setUp() {
        p=ParticipantsInDB.getInstance();
    }
    void tearDown() {
        ParticipantsInDB.freeInstance();
    }
    void testAdd() {
        p.deleteAllParticipants();
        p.addParticipant(part1);
        assertEquals(p.getCount(), 1);
    }
    void testAdd2() {
        p.deleteAllParticipants();
        p.addParticipant(part1);
        p.addParticipant(part2);
        assertEquals(p.getCount(), 2);
    }
    void testEnum() {
        p.deleteAllParticipants();
        p.addParticipant(part2);
        p.addParticipant(part1);
        ParticipantEnumeratorById penum=new ParticipantEnumeratorById();
        assertTrue(penum.next());
        assertTrue(penum.get().equals(part1));
        assertTrue(penum.next());
        assertTrue(penum.get().equals(part2));
        assertTrue(!penum.next());
        penum.close();
    }
}

```

5. Point out and remove the duplication in the code:

```

class ReportCatalogueIndexCommandParser {
    final String NO_GROUPING = "orgNoGrouping";
    static final int ORG_CATALOG = 0;
    static final int PART_CATALOG = 1;
    int getGroupingType(String grouping) {
        if (grouping.equals(NO_GROUPING)) {
            return ORG_CATALOG;
        } else if (grouping.equals("orgGroupByCountry")) {
            return ORG_CATALOG;
        } else if (grouping.equals("orgGroupByTypeOfOrgName")) {
            return ORG_CATALOG;
        } else if (grouping.equals("part")) {
            return PART_CATALOG;
        }
    }
}

```

Licensed for viewing only. Printing is prohibited. For hard copies, please purchase from www.agileskills.org

```
    } else
        throw new IllegalArgumentException("Invalid grouping!");
    }
    ...
}
```

Each branch of if-then is doing a similar thing. Extract the strings used in the conditions into a set.

```
class ReportCatalogueIndexCommandParser {
    final String NO_GROUPING = "orgNoGrouping";
    static final int ORG_CATALOG = 0;
    static final int PART_CATALOG = 1;

    int getGroupingType(String grouping) {
        Set orgGroupings = new TreeSet();
        orgGroupings.add(NO_GROUPING);
        orgGroupings.add("orgGroupByCountry");
        orgGroupings.add("orgGroupByTypeOfOrgName");
        if (orgGroupings.contains(grouping)) {
            return ORG_CATALOG;
        }
        if (grouping.equals("part")) {
            return PART_CATALOG;
        } else
            throw new IllegalArgumentException("Invalid grouping!");
    }
}
```

Using a set may be an overkill. A simpler way is to just extract the return statement:

```
class ReportCatalogueIndexCommandParser {
    final String NO_GROUPING = "orgNoGrouping";
    static final int ORG_CATALOG = 0;
    static final int PART_CATALOG = 1;
    int getGroupingType(String grouping) {
        if (grouping.equals(NO_GROUPING) ||
            grouping.equals("orgGroupByCountry") ||
            grouping.equals("orgGroupByTypeOfOrgName")) {
            return ORG_CATALOG;
        } else if (grouping.equals("part")) {
            return PART_CATALOG;
        } else
            throw new IllegalArgumentException("Invalid grouping!");
    }
    ...
}
```

6. Point out and remove the duplication in the code:

```
class Restaurant extends Account {
    //the string "Rest" is concated with the restaurant ID to
    //form the key.
    final static String RestaurantIDText = "Rest";
    String website;
    //address in Chinese.
    String addr_cn;
```

```

//address in English.
String addr_en;
//the restaurant would like to update its fax # with this. After it is
//confirmed, it will be stored in Account. Before that, it is stored
//here.
String numb_newfax;
boolean has_NewFax = false;
//a list of holidays.
Vector Holiday; // a holiday
//id of the category this restaurant belongs to.
String catId;
//a list of business session. Each business session is an array
//of two times. The first time is the start time. The second time
//is the end time. The restaurant is open for business in each
//session.
Vector BsHour; //Business hour
...
//y: year.
//m: month.
//d: date.
void addHoliday(int y,int m,int d){
    if(y<1900) y+=1900;
    Calendar aHoliday = (new GregorianCalendar(y,m,d,0,0,0));
    Holiday.add(aHoliday);
}
public boolean addBsHour(int fromHr, int fromMin, int toHr, int toMin){
    int fMin = fromHr*60 + fromMin; //start time in minutes.
    int tMin = toHr*60 + toMin; //end time in minutes.
    //make sure both times are valid and the start time is earlier
    //than the end time.
    if(fMin > 0 && fMin <= 1440 && tMin > 0 && tMin <=1440 && fMin < tMin){
        GregorianCalendar bs[] = {
            new GregorianCalendar(1900,1,1, fromHr, fromMin,0),
            new GregorianCalendar(1900,1,1, toHr, toMin,0)
        };
        BsHour.add(bs);
        return true;
    } else {
        return false;
    }
}
}

```

The code to convert from hours plus minutes into just minutes is duplicated. The code to validate the number of minutes is duplicated. The code to create a GregorianCalendar on January 1 in year 1900 is duplicated.

```

class Restaurant extends Account {
    ...
    int getMinutesFromMidNight(int hours, int minutes) {
        return hours*60+minutes;
    }
    boolean isMinutesWithinOneDay(int minutes) {
        return minutes>0 && minutes<=1440;
    }
    GregorianCalendar convertTimeToDate(int hours, int minutes) {
        return new GregorianCalendar(1900, 1, 1, hours, minutes, 0);
    }
}

```

```

public boolean addBsHour(int fromHr, int fromMin, int toHr, int toMin){
    int fMin = getMinutesFromMidNight(fromHr, fromMin);
    int tMin = getMinutesFromMidNight(toHr, toMin);
    if (isMinutesWithinOneDay(fMin) &&
        isMinutesWithinOneDay(tMin) &&
        fMin < tMin){
        GregorianCalendar bs[] = {
            convertTimeToDate(fromHr, fromMin),
            convertTimeToDate(toHr, toMin)
        };
        BsHour.add(bs);
        return true;
    } else {
        return false;
    }
}
}

```

7. Point out and remove the duplication in the code:

```

class Order {
    ...
    boolean IsSameString(String s1, String s2){
        if(s1==s2) return true;
        if(s1==null) return false;
        return(s1.equals(s2));
    }
}
class Mail {
    ...
    static boolean IsSameString(String s1, String s2) {
        if (s1 == s2)
            return true;
        if (s1 == null)
            return false;
        return (s1.equals(s2));
    }
}

```

The IsSameString method is duplicated.

```

class StringComparer {
    static boolean IsSameString(String s1, String s2) {
        if (s1 == s2)
            return true;
        if (s1 == null)
            return false;
        return (s1.equals(s2));
    }
}
class Order {
    ...
    //If possible, remove this method and let the other code call
    //StringComparer.IsSameString directly.
    boolean IsSameString(String s1, String s2){
        return StringComparer.IsSameString(s1, s2);
    }
}

```

```
class Mail {
    ...
    //If possible, remove this method and let the other code call
    //StringComparer.IsSameString directly.
    static boolean IsSameString(String s1, String s2) {
        return StringComparer.IsSameString(s1, s2);
    }
}
```

8. Point out and remove the duplication in the code:

```
class FoodDB {
    //find all foods whose names contain both the keywords. returns
    //an iterator on these foods.
    public Iterator srchFood(String cName, String eName){
        //it contains all the foods to be returned.
        TreeMap foodTree = new TreeMap();
        Iterator foodList;
        Food food;
        foodList = getAllRecords();
        while (foodList.hasNext()){
            food = (Food) foodList.next();
            //do its names contain both keywords?
            if ((cName==null || //null or "" means no key is specified
                cName.equals("") ||
                food.getCName().indexOf(cName)!=-1)
                &&
                (eName==null || //null or "" means no key is specified
                eName.equals("") ||
                food.getEName().indexOf(eName)!=-1)){
                foodTree.put(food.getFoodKey(), food);
            }
        }
        return foodTree.values().iterator();
    }
}
```

The checking of whether the name contains a keyword is duplicated.

```
class FoodDB {
    boolean nameMatchKeyword(String name, String keyword) {
        return keyword==null ||
            keyword.equals("") ||
            name.indexOf(keyword)!=-1;
    }
    public Iterator srchFood(String cName, String eName){
        TreeMap foodTree = new TreeMap();
        Iterator foodList;
        Food food;
        foodList = getAllRecords();
        while (foodList.hasNext()){
            food = (Food) foodList.next();
            if (nameMatchKeyword(food.getCName(), cName)
                &&
                nameMatchKeyword(food.getEName(), eName)) {
                foodTree.put(food.getFoodKey(), food);
            }
        }
    }
}
```

```

        return foodTree.values().iterator();
    }
}

```

9. Point out and remove the duplication in the code:

```

class UIDialogCustomerMain extends JDialog {
    JButton btnOrderDel;
    JButton btnCustChangePassword;
    void bindEvents() {
        ...
        btnOrderDel.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Dialog d = new UIDialogCustomerDeleteOrder(UIDialogCustomerMain.this,
"Del Order", true);
                d.pack();
                d.setLocation(400,200);
                d.setVisible(true);
            }
        });
        btnCustChangePassword.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Dialog d = new UIChangeAccountPW(UIDialogCustomerMain.this, "chg pw",
true);
                d.pack();
                d.setLocation(400,200);
                d.setVisible(true);
            }
        });
    }
    ...
}

class UIDialogRestaurantMain extends JDialog {
    JButton btnChangePassword;
    void bindEvents() {
        ...
        btnChangePassword.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Dialog d = new UIChangeAccountPW(UIDialogRestaurantMain.this, "chg
pw", true);
                d.pack();
                d.setLocation(400,200);
                d.setVisible(true);
            }
        });
    }
    ...
}

```

The code packing the dialog, setting its location and showing it is duplicated. The change password button is also duplicated.

```

class UIDialogShower {
    public static void show(Dialog d, int left, int top) {
        d.pack();
        d.setLocation(left, top);
        d.setVisible(true);
    }
}

```

```

    public static void showAtDefaultPosition(Dialog d) {
        show(d, 400, 200);
    }
}
class ChangePasswordButton extends JButton {
    public ChangePasswordButton(final JDialog enclosingDialog) {
        addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Dialog d = new UIChangeAccountPW(enclosingDialog, "chg pw", true);
                UIDialogShower.showAtDefaultPosition(d);
            }
        });
    }
}
class UIDialogCustomerMain extends JDialog {
    JButton btnOrderDel;
    ChangePasswordButton btnCustChangePassword = new ChangePasswordButton(this);
    void bindEvents() {
        ...
        btnOrderDel.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Dialog d = new UIDialogCustomerDeleteOrder
                    (UIDialogCustomerMain.this, "Del Order", true);
                UIDialogShower.showAtDefaultPosition(d);
            }
        });
    }
    ...
}
class UIDialogRestaurantMain extends JDialog {
    ChangePasswordButton btnChangePassword = new ChangePasswordButton(this);
    void bindEvents() {
        ...
    }
    ...
}

```

10. Suppose that there are two kinds of rentals: book and movie. Point out and remove the duplication in the code:

```

public class BookRental {
    private String bookTitle;
    private String author;
    private Date rentDate;
    private Date dueDate;
    private double rentalFee;
    public boolean isOverdue() {
        Date now=new Date();
        return dueDate.before(now);
    }
    public double getTotalFee() {
        return isOverdue() ? 1.2*rentalFee : rentalFee;
    }
}
public class MovieRental {
    private String movieTitle;
    private int classification;
    private Date rentDate;

```

```
private Date dueDate;
private double rentalFee;
public boolean isOverdue() {
    Date now=new Date();
    return dueDate.before(now);
}
public double getTotalFee() {
    return isOverdue() ? 1.3*rentalFee : rentalFee;
}
}
```

Some fields such as rentDate, dueDate and etc. are duplicated. The isOverdue method is duplicated. The structure of the getTotalFee method is duplicated.

```
public abstract class Rental {
    private Date rentDate;
    private Date dueDate;
    private double rentalFee;
    protected abstract double getOverduePenaltyRate();
    public boolean isOverdue() {
        Date now=new Date();
        return dueDate.before(now);
    }
    public double getTotalFee() {
        return isOverdue() ? getOverduePenaltyRate()*rentalFee : rentalFee;
    }
}
public class BookRental extends Rental {
    private String bookTitle;
    private String author;
    protected double getOverduePenaltyRate() {
        return 1.2;
    }
}
public class MovieRental extends Rental {
    private String movieTitle;
    private int classification;
    protected double getOverduePenaltyRate() {
        return 1.3;
    }
}
```