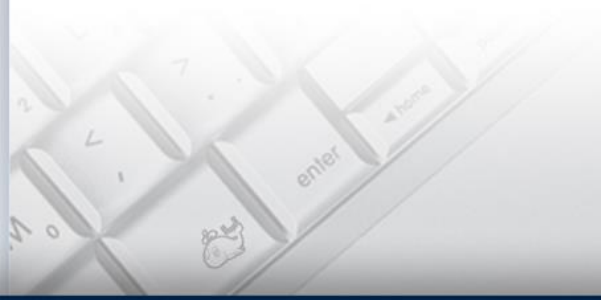
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Антипаттерны проектирования: как не нужно писать приложения

Евгений Беркунский, НУК
eberkunsky@gmail.com
ноябрь, 2012

по материалам
Андрея Дмитриева
Andrei.Dmitriev@Sun.Com



Цель

- Изучить наиболее распространенные антипаттерны проектирования, их симптомы и способы исправления.
- Изучить пути перекрытия уязвимостей путем отказа от использования небезопасных паттернов.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Что такое уязвимость?

Место в системе, которое позволяет атакующему

- нарушить целостность
- нарушить конфиденциальность,
- обойти средства контроля доступа,
- повредить доступности сервисов,
- нарушить соответствие данных друг другу,
- отследить механизм работы системы,
- получить доступ к данным и процессам системы.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Почему появляются уязвимости?

1. Просчеты при проектировании архитектуры.
2. Ошибки при настройке.
3. Неправильная логика.
4. Небезопасный стиль программирования (антипаттерны).

Презентация охватывает антипаттерны.

Антипаттерны

- Антиподы паттернов проектирования.
- Практики программирования, которых рекомендуется по возможности избегать.
- Могут существовать в
  - клиентском коде
  - библиотеках
  - Java библиотеках

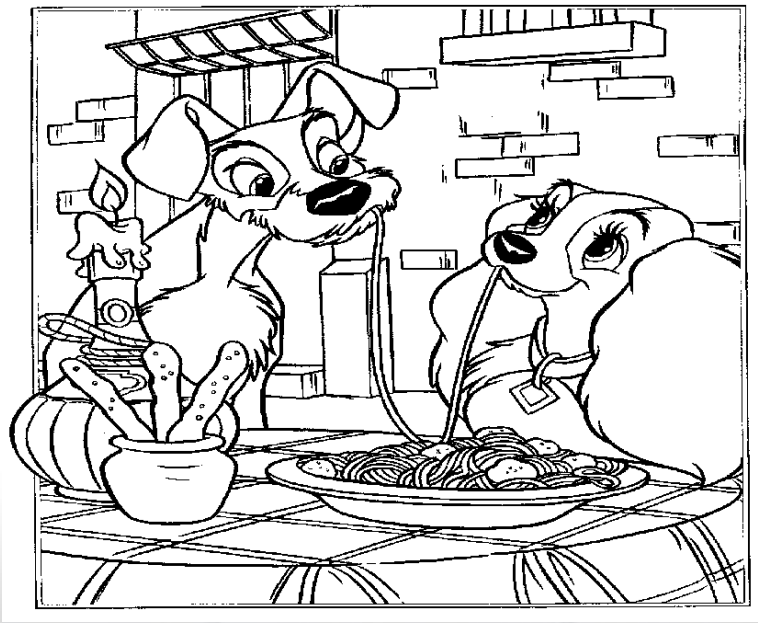
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Стилистические антипаттерны

- «Спагетти».
- Дублирование кода.
- «Вшивание» данных (hardcoding).
- Программирование методом подбора.
- Раздувание класса.
- Использование исключений не по назначению.

«Спагетти»

- Характерны функции очень большого размера.
- Появляется при оценке качества программы с позиции — «Программа работает».



«Спагетти»: решение

- Оформлять программу так, чтобы размер функций не превышал десяти-двадцати строк.
- Использовать метрики для обнаружения "спагетти"-функций (методов).
- При разрастании функции выделять связные куски кода в отдельные методы.

Дублирование кода

Методом копирования текста программы, на основе однажды написанного кода можно создавать блоки, выполняющие схожие друг с другом задачи.

Дублирование кода: диагноз

- Симптомы: программа после изменения в некоторых случаях ведет себя также как и раньше.
- Возможная причина: были изменены не все места, где используется похожий код.
- Лечение:
  - рефакторинг «выделение метода».

«Вшивание» данных (hardcoding)

- Жесткое «вшивание» в программный код различных данных, касающихся окружения программы:
  - путь к файлу конфигурации
  - имя почтового сервера
  - название таблицы БД
  - и т.д.
- Программа может работать только в определенных условиях (на компьютере разработчика).
- Данный антипаттерн невозможно выявить на машине разработчика без досконального изучения кода.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

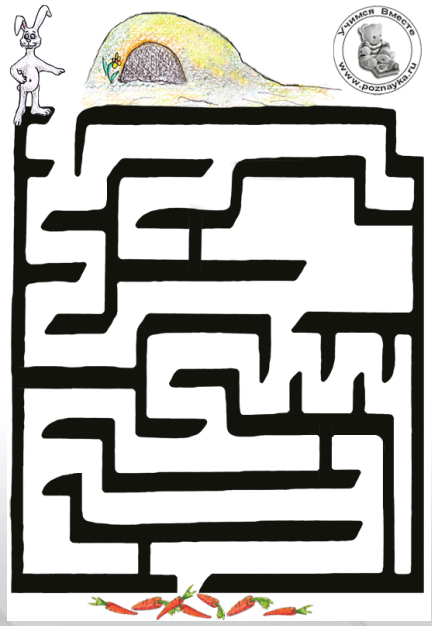
«Вшивание» данных: решение

- Избегать жесткого занесения каких-либо значений в программу.
- Использовать именованные константы.
- Возможно передавать значения как параметры, завести файл конфигурации или системное свойство.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Программирование методом подбора

- Программирование с использованием случайного поиска решения.
- Подбирать можно:
  - Наличие и порядок вызовов функций,
  - Значения параметров,
  - и т.д.



Программирование методом подбора

- Данный подход устраняет только видимую часть ошибки и не дает программисту понимания сути происходящего.
- Решение: проанализировать проблемную ситуацию и найти действительную причину.

Раздувание класса

- Класс, на который возложено слишком много обязанностей.
- Большие классы тяжело поддерживать, они очень неповоротливы и не любят рефакторинг.
- Решение: каждый класс должен иметь одно конкретное назначение, которое м[illegible] несколькими словами.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Использование исключений не по назначению

- Представляет собой реализацию нормальной логики работы программы с помощью механизма исключений:
  - выход из блока с помощью брошенного исключения.
- Использование исключений как ин управления логикой программы вн неоднозначность.
- Решение: использовать исключени с единственной целью — проинформировать систему об оши

Общие антипаттерны

- Сломанный диспатч.
- Некорректные данные.
- Тип-самозванец.
- Фиктивная реализация.
- Недостаточная инициализация.

Сломанный диспатч

- Есть класс, предоставляющий сервис и класс-клиент, его использующий.
- При внесении изменений в один из классов, реализующих сервис, клиент начинает работать неверно.
- На первый взгляд, затронутая функциональность не связана с той, которая перестала рабс

Сломанный диспатч: диагноз

- Симптомы: программа перестала вести себя по-прежнему при внесении изменений в несвязный компонент.
- Возможная причина: при переопределении метода в классе-наследнике, клиент в соответствии с принципом полиморфизма вызывает новый метод, вместо старого.
- Лечение:
  - преобразование типов,
  - пересмотр набора методов для данного сервиса.

Некорректные данные

- Во время работы программы происходят ошибки в недетерминированный момент времени.
- При выполнении некоторой конкретной работы приложение работает неправильно.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Некорректные данные: диагноз

- Симптомы: при определенном стечении обстоятельств (точные причины могут быть неизвестны), приложение работает неверно.
- Возможная причина: приложение пытается использовать некорректные данные, пропущенные на более ранних этапах программы.
- Лечение:
  - Проверка данных как можно раньше по времени,
  - Проверка входных параметров методов.

Тип-самозванец

- Класс выбирает путь исполнения операторов в зависимости от значения своего поля.
- Приложение не использует статическую типизацию классов для выбора пути исполнения.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Тип-самозванец: пример

Класс Form использует строку для выбора алгоритма вычисления площади.

```
public class Form{
    String shape;
    double scale;
    public Form(String shape, double scale){…}
    public double getArea(){
      if  shape.equals(“square”))
        return scale*scale;
      } else if  shape.equals(“circle”)
        return Math PI*scale*scale
      } else  return 0.0;}
    }
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Тип-самозванец: пример (cont.)

- Что произойдет при вызове:

```
Form f = new Form(“sqaure”); ?
```

- Как добавлять новые типы форм в программу?

Тип-самозванец : диагноз

- Симптомы: программа одинаково обрабатывает принципиально различные данные.
- Возможная причина: для различных типов данных в программе используются теговые поля.
- Лечение: разделение отличных типов данных на классы.

Тип-самозванец: правильное решение

Использование различных классов для различных сущностей:

```
public abstract class Form {
   double scale;
   public abstract double getArea();
}

class Square extends Form {
 public double getArea(){
   return scale*scale;
 }
}

class Circle extends Form {
 public double getArea(){
   return Math.PI*scale*scale;
 }
}
```

Фиктивная реализация

- Интерфейс формализует сигнатуру методов.
- Кроме этого могут существовать другие правила, которым программа должна следовать.
- Наблюдается недостаток языковых средств для формализации всех отношений.

Ложный опёнок серно-жёлтый.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Фиктивная реализация: пример

```
interface Stack {
 public Object pop();
 public void push(Object o);
 public boolean isEmpty();
}
```

Как описать пункты:

1. Если объект помещается в стек, то следующий вызов pop() должен его вернуть.
2. Если isEmpty()==true, то обращение к методу pop() должно вызвать исключение.
3. Как выполнять операции на нескольких потоках?

Фиктивная реализация: диагноз

- Симптомы: клиентский класс, использующий конкретную реализацию интерфейса, ломается.
- Возможная причина: реализация интерфейса не удовлетворяет некоторых ожиданий клиента.
- Лечение:
  - исправить реализацию чтобы она удовлетворяла необходимым правилам,
  - Явно указать все правила в документации,
  - Покрыть случаи тестами.

Недостаточная инициализация

- Построение экземпляра класса может проходить в несколько этапов (операторов или вызовов методов).
- Невыполнение некоторых этапов может происходить по причине:
  - Отсутствия инициализирующего кода,
  - Отложенной инициализации,
  - Исключения.
- Пропущенные этапы могут не влиять на процесс создания, но объект остается в незавершенном состоянии.

Недостаточная инициализация: диагноз

- Симптомы: `NullPointerException`, неправильные результаты.
- Возможная причина: конструктор не инициализировал все поля.
- Лечение:
  - Правильная инициализация,
  - Использование значений по умолчанию (вместо null и т.п.),
  - Использовать метод isInitialized(),
  - Включение дополнительных (корректных) конструкторов,
  - Переписать класс.

Другие антипаттерны

- Использование интерфейса как хранилища констант.
- Несогласованное удаление объекта.

Использование интерфейса как хранилища констант

- Может существовать набор значений, уникальных для всей системы.
- Вариант сделать их доступными для всех заинтересованных клиентов:
  - Поместить необходимые константы в интерфейс,
  - Реализовывать этот интерфейс заинтересованным клас

Использование интерфейса как хранилища констант: пример

```
public interface XConstants {
    static final int X_PROTOCOL = 11 ;
    public static final int X_PROTOCOL_REVISION = 0;
}

public class XBaseWindow implements XConstants {
  //использование констант без префикса
}
```

Использование интерфейса как хранилища констант: недостатки

Класс, помимо нужных получает и ненужные константы

- в область видимости
- и в качестве API.

Хранилище констант: правильное решение

- Глобальный класс(-одиночка), предоставляющий необходимые константы.
- Использование static import (с версии JDK1.5.0).

Несогласованное удаление объекта

- Симптомы: программа некорректно управляет ресурсами, не удаляя их или удаляя их несколько раз.
- Возможная причина: на некоторых путях программы ресурсы не освобождаются один раз.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Несогласованное удаление: пример

```
class TableWalker {
    Connection conn = …; //получаем соединение с БД.
    …
 public void walk() throws SQLException {
    Statement st = conn.createStatement();
    ResultSet rs = st.executeQuery("SELECT * FROM student");
    while (rs.next()) { execute(rs); }
        conn.close();
 }
}
```

Несогласованное удаление: пример (cont.)

```
class TablePrinter extends TableWalker{
   …
 public void execute(ResultSet rs) throws SQLException {
    String s = rs.getString("FIRST_NAME");
    System.out.println(s);
    …
    //При инициировании исключения происходит выход из
    //данного метода и из метода walk() суперкласса.
    // Переменная conn не освобождается.
    return;
 }
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Несогласованное удаление объекта: правильное решение

- Переместить участок программы, управляющий ресурсами в тот же метод, где ресурс создается.
- Предусмотреть все пути программы.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Опасные антипаттерны: C и Java

- C:
  - Уязвимости часто связаны с переполнением буфера.
- Java:
  - JRE управляет памятью:
    - Проверка индекса массива
    - Отсутствие арифметики указателей
  - JRE часто исполняет непроверенный код.
    - Нужно предотвращать несанкционированный доступ к ресурсам
- Результат: различные антипаттерны для языков C и Java.

Опасные антипаттерны

- Предположение о том, что объект неизменяем.
- Доверие к чужому коду.
- Некорректное использование public static переменных.
- Вера в то, что исключение не позволяет создать объект.
- Уверенность, что исключения безвредны.
- Восстановление объекта.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Предположение о том, что объект неизменяем

Метод данного класса, возвращающий массив, может предоставить клиенту возможность изменять свое содержимому.

Как?

```
public class Test {
   private Object [] array = {new Object(),
           new Object()};
   public Object [] getArray() {
           return array;
   }
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Предположение о том, что объект неизменяем: уязвимость

Атакующий может помещать в массив свои экземпляры объектов:

```
test.getArray()[0] = new Object();
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Предположение о том, что объект неизменяем: проблемы

- Данные любых объектов могут быть изменены
- Изменения могут приводить к краху системы.
- Изменения в данных, обеспечивающих контроль безопасности может дать несанкционированный доступ.

Предположение о том, что объект неизменяем: правильное решение

```
public class Test {
   private Object [] array = {new Object(), new Object()};
   public Object [] getArray()    {
     return java.util.Arrays.copy(array, array.length);
   }
}
```

Доверие к чужому коду

- Некорректные операции, выполняемые чужим кодом могут нарушить работу программы.
- Данный метод может предоставить доступ практически к любому файлу, игнорируя права пользователя. Как?

```
public RandomAccessFile openFile(final java.io.File f)
{
  askUserPermission(f.getPath());
   return(RandomAccessFile)AccessController.doPrivileged()
    {
        public Object run(){
               return new RandomAccessFile(f.getPath());
        }
   }
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Доверие к чужому коду: атака

Класс переопределяет метод getPath():

```
public class BadFile extends java.io.File {
      private int count;
      public String getPath(){
        return (++count==1)?
          "/tmp/foo":"/etc/passwd";
    }
}
```

Доверие к чужому коду: проблема

- Проверка может быть успешно обойдена, если она проверяет данные, которые может контролировать атакующий.
- Библиотечные классы тоже могут быть небезопасными.
- Классы без модификатора final могут иметь наследников с переопределенными методами.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Доверие к чужому коду: решение

- Не предполагайте что входные данные неизменяемы.
- Копируйте данные и проводите проверки над ними.

```
public RandomAccessFile openFile(File f){
    final File copy =f.clone();
    askUserPermission(copy.getPath());
    ...
    return new RandomAccessFile(copy.getPath());
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Доверие к чужому коду: правильное решение

- Метод clone() создает копию атакующего класса.

```
public RandomAccessFile openFile(java.io.File f){
final java.io.File copy =f.clone();
 askUserPermission(copy.getPath());
    ...
}
```

- Правильнее будет копировать немодифицируемые данные (String) и предоставлять доступ на их основе:

```
java.io.File copy =new java.io.File(f.getPath());
```

Некорректное использование public static переменных

Класс содержит публичную переменную.

```
public class FunctionTable {
    public static FuncLoader m_functions;
}
```

Некорректное использование public static переменных:атака

Переменная просто может быть заменена на другую.

```
FunctionTable.m_functions =<new_table>;
```

Некорректное использование public static переменных:проблема

- Состояние класса может быть изменено неавторизованным кодом.
- Статические переменные являются глобальными для всего рабочего окружения Java и могут быть использованы для общения между частями системы.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Некорректное использование public static переменных: правильное решение (1/2)

- Уменьшить область видимости:

```
static FuncLoader m_functions;
```

```
private static FuncLoader m_functions;
```

- Использовать public static только для констант.
- По возможности использовать enum.
- Сделать public static поля final (значение final поля не может быть изменено):

```
public class MyClass {
    public static final int LEFT =1;
    public static final int RIGHT =2;
}
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Некорректное использование public static переменных: правильное решение (2/2)

- Определить методы для доступа к изменяемым mutable static полям.
- Добавить необходимые проверки.

```
public class MyClass {
    private static byte [] data;;
    public static byte [] getData((){
        return data.clone();
    }
    public static void setData(byte [] b)){
        securityCheck();
        data =b.clone();
    }
}
```

Вера в то, что исключение не позволяет создать объект

Объект не должен существовать если проверка на безопасность не была пройдена.

```
public class ClassLoader {
    public ClassLoader(){
        securityCheck();
        init();
    }
}
```

Вера в то, что исключение не позволяет создать объект: атака

Переопределение метода finalize() позволяет воссоздать объект, на который не существует ссылки.

```
public class MyCL extends ClassLoader {
    static ClassLoader cl;

    protected void finalize(){ cl =this; }
    public static void main(String [] s){
        try { new MyCL();
        }catch (SecurityException e){…}
        System.gc();
        System.runFinalization();
        System.out.println(cl);
    }
}
```

Вера в то, что исключение не позволяет создать объект: проблема

- Бросаемое исключение не гарантирует что объект не будет восстановлен.
- Вызов конструктором метода, бросающего исключение, все равно позволяет воссоздать объект.

Вера в то, что исключение не позволяет создать объект: правильное решение

- По возможности описывать класс как final.
- Если метод finalize() может быть переопределен, сделать так, чтобы частично инициализированный объект был нефункционален.
- Не выставлять значения полей до тех пор пока все проверки не пройдены.
- Использовать флаг завершения инициализации.

```
public class ClassLoader {
    private boolean initialized =false;
    ClassLoader(){
      securityCheck();
      init();
      initialized =true;
      //check flag in all relevant methods
    }
}
```

Уверенность в том, что исключения безвредны

Исключения могут содержать данные, выдающие информацию о пользователе (например, имя пути к файлу может содержать имя пользователя).

```
public class PersonalData {
    public load() throws IOException {
        String homedir =System.getProperty("user.dir ");
        File f =new File
            (homedir,"personal.dat ");
        FileInputStream s =
            new FileInputStream(f);
    }
}
```

Уверенность в том, что исключения безвредны: атака

Атакующий может попробовать выяснить важную информацию.

```
try {
    personal.load();
}catch (IOException e){
    String homedir =parsePath(e.message());
    String username =parseUser(homedir);
 }
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Уверенность, что исключения безвредны: правильное решение

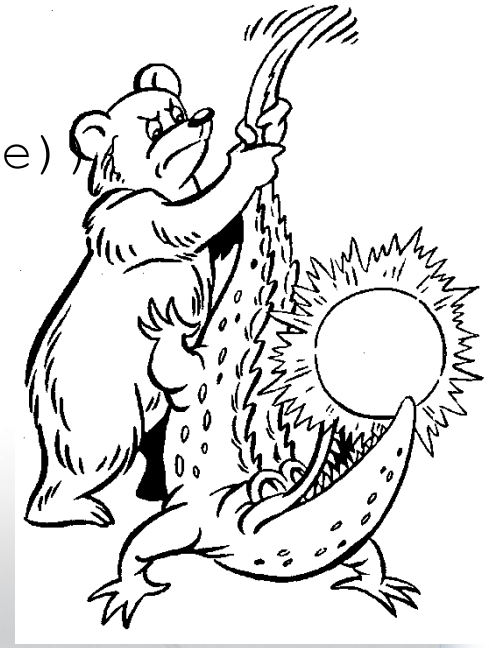
Скрывайте информацию, задавая нейтральные значения полей или заменяя исключения.

```
public class PersonalData {
    public load()throws IOException {
        try {
            ...
        }catch (Exception e){
            throw new IOException("Could not load data ");
        }
    }
}
```

Восстановление объекта

- Класс проверяет значение на допустимость в конструкторе.
- Если входной параметр не удовлетворяет условию, то создается исключение.
- Как следствие, выполнение оператора new прерывается и объект не создается.

```
public class BigInteger extends Number {
    private int signum;
    public BigInteger(int signum,byte [] magnitude)
          if (signum <-1 ||signum >1){
              throw new NumberFormatException();
          }
          ...
    }
}
```



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Восстановление объекта: атака

Атакующий может создать объект посредством десериализации ложного объекта.

```
ObjectInputStream is = new
   FileInputStream(“badfile.ser”);
BigInteger bigInt = is.readObject();
```

Восстановление объекта: проблема

- При десериализации конструктор не выполняется.
- Атакующий может создать сериализованный объект с неверными значениями полей.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Восстановление объекта: правильное решение

Создать свой метод readObject() и проводить в нем те же самые проверки, что и в конструкторе.

```
private void readObject(ObjectInputStream s){
    s.defaultReadObject();
    //Validate signum
    if (signum <-1 ||signum >1)
        throw new StreamCorruptedException();
 }
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Выводы

- Существуют классы ошибок, часто встречающихся при проектировании и написании программ.
- Использование антипаттернов делает систему уязвимой.
- Проявления неудачных паттернов достаточно широки.
- Нахождение неудачного паттерна позволяет улучшить программу.
- Существуют пути избавления от антипаттернов.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Ссылки

- Путеводитель по безопасному программированию:
  - http://java.sun.com/security/seccodeguide.html
- Bug patterns in Java, Eric Allen
  - http://www.cs.rice.edu/~eallen/
- Собрание публикаций по теме:
  - http://www.antipatterns.com/briefing/index.htm
  - http://en.wikipedia.org/wiki/Anti-pattern
  - http://www.insidecpp.ru/antipatterns/
  - http://c2.com/cgi/wiki?AntiPatternsBook
- Antipatterns: Identification, Refactoring and Management, Laplante, Phillip A. and Colin J. Neill.
- *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, Brown, William J.; Raphael C. Malveau.

Вопросы?

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Антипаттерны проектирования: как не нужно писать приложения

Евгений Беркунский, НУК
eberkunsky@gmail.com
ноябрь, 2012

по материалам
Андрея Дмитриева
Andrei.Dmitriev@Sun.Com