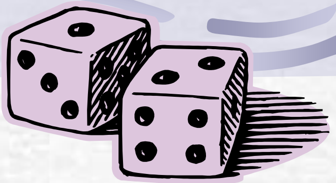
Refactoring

A Small-Scale Example

A Small-Scale Example



- A method of some dice game class that throws a couple of dice and returns a result.
- 'dice' is an array of 'Die' objects.

```
public int getScore()
{
        int result;
        result = (int)(Math.random() * 6) + 1;
        dice[0].setFaceValue(result);
        result = (int)(Math.random() * 6) + 1;
        dice[1].setFaceValue(result);
        int score = dice[0].getFaceValue() + dice[1].getFaceValue();
        return score;
}
```

We don't want to directly access the internal variable *dice*...

Refactoring Step 1 :: Encapsulate Field

- Use accessor methods!
- Do not directly access an object´s fields within its methods.

We see duplicate code...

```
public int getScore()
{
        int result;
        result = (int)(Math.random() * 6) + 1;
        getDie(0).setFaceValue(result);
        result = (int)(Math.random() * 6) + 1;
        getDie(1).setFaceValue(result);
        int score =getDie(0).getFaceValue()
                    +getDie(1).getFaceValue();
        return score;
}
```

Refactoring Step 2 :: Extract Method

- To reduce duplicate code we extract a new method

Does this name make sense ..? Is it obvious what is or should be occurring ...?

```
public int getScore() {
    int result;
    result = rollDie();
    getDie(0).setFaceValue(result);
    result = rollDie();
    getDie(1).setFaceValue(result);
    int score = getDie(0).getFaceValue()+getDie(1).getFaceValue();
    return score;
}

public int rollDie() {
    return (int)(Math.random() * 6) + 1;
}
```

Step 3 ::Rename Method

- Change names to be more meaningful.

Why are we using a temporary variable?

```
public int throwDice() {
    int result;
    result = rollDie();
    getDie(0).setFaceValue(result);
    result = rollDie();
    getDie(1).setFaceValue(result);
    int score = getDie(0).getFaceValue()+getDie(1).getFaceValue();
    return score;
}

public int rollDie() {
    return (int)(Math.random() * 6) + 1;
}
```

4 :: Replace Temp with Query

- Use a query method instead of a temporary variable.

```
public int throwDice(){
   int result;
   result = rollDie();
   getDie(0).setFaceValue(result);
   result = rollDie();
   getDie(1).setFaceValue(result);
   return getDiceValue();
}

public int rollDie() { return (int)(Math.random() * 6) + 1; }

int getDiceValue() {
   return getDie(0).getFaceValue() + getDie(1).getFaceValue();
}
```

Why isn’t this a part of the *Die* object?

5 :: Move Method & Rename Method

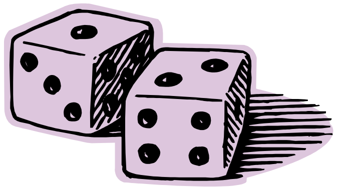
- Dice objects are data objects.
- It would be better to move the rollDie() method to the Die class and have this method set the state of the object.
- The rollDie() method can also be renamed to roll().

```
public class Die {
//…

    public void roll() {
        setFaceValue((int)(Math.random() * 6) + 1);
    }
}
```

5 ::Move Method (continued)

- Update original the code to reflect the move & name change

```
public int throwDice() {
    getDie(0).roll();
    getDie(1).roll();
    return getDiceValue();
}

int getDiceValue() {
    return getDie(0).getFaceValue()+getDie(1).getFaceValue();
}
```



The code is beginning to look much cleaner.

Let’s kill those evil local variables

Example

how to calculate the ISO week number for a given date

http://en.wikipedia.org/wiki/ISO_week_date

```
public class WeekCalculator {

private DateTime GetIsoWeekOne(int year) {
// get the date for the 4-Jan for this year
DateTime dt = new DateTime(year, 1, 4);
// get the ISO day number for this date 1==Monday, 7==Sunday
int dayNumber = (int)dt.DayOfWeek; // 0==Sunday, 6==Saturday
if (dayNumber == 0) { dayNumber = 7;}

// return the date of the Monday that is less than or equal
// to this date
return dt.AddDays(1 - dayNumber);}
public int GetIsoWeek(DateTime dt) {
DateTime week1;
int IsoYear = dt.Year;
if (dt >= new DateTime(IsoYear, 12, 29)) {
week1 = GetIsoWeekOne(IsoYear + 1);
if (dt < week1) {
week1 = GetIsoWeekOne(IsoYear);}
else {IsoYear++;}}
else {
week1 = GetIsoWeekOne(IsoYear);
if (dt < week1) {
week1 = GetIsoWeekOne(--IsoYear);}}
return (IsoYear * 100) + ((dt - week1).Days / 7 + 1);}}
```

Lets start

I can see *Extract Method* screaming at me from all the comments in the GetIsoWeekOne() method. This eliminates dayNumber

```
private int GetIsoDayNumber(DateTime date) {
if (date.DayOfWeek = DayOfWeek.Sunday)
    return 7;
return (int)date.DayOfWeek;
}
```

--- aaaarrrrggghhhh, a Cast, one thing at a time.

```
private int GetIsoDayNumber(DateTime date) {
if (date.DayOfWeek == DayOfWeek.Sunday)
return 7;
return (int)date.DayOfWeek;
}
private DateTime GetIsoWeekOne(int year) {
// get the date for the 4-Jan for this year
DateTime dt = new DateTime(year, 1, 4);
// return the date of the Monday that is less than or equal
// to this date
return dt.AddDays(1 - GetIsoDayNumber(dt));}
```

That return Statement

Its about Mondays!

Can’t see any Mondays in the code … so …

```
private DateTime GetPreviousMonday(DateTime
date) {
return date.AddDays(1-GetIsoDayNumber(date));
}
```

Looking good ….almost

```
private  DateTime GetPreviousMonday(DateTime date) {
return date.AddDays(1 - GetIsoDayNumber(date));}

private DateTime GetIsoWeekOne(int year) {
// get the date for the 4-Jan for this year
DateTime dt = new DateTime(year, 1, 4);
return GetPreviousMonday(dt);}
```

But good grief a temporary variable – really!

get rid of the temp completely

```
private int GetIsoDayNumber(DateTime date) {
if (date.DayOfWeek == DayOfWeek.Sunday)
return 7;
return (int)date.DayOfWeek;}

private DateTime GetPreviousMonday(DateTime date) {
return date.AddDays(1 - GetIsoDayNumber(date));}

private DateTime Get4thOfJanuary(int year) {
return new DateTime(year, 1, 4);}

private DateTime GetIsoWeekOne(int year) {
return GetPreviousMonday(Get4thOfJanuary(year));}
```

GetIsoWeek()

- Here there are two temps (week1, IsoYear)

- In fact IsoYear is that most malignant of all temps: it's mutable (IsoYear++ and --IsoYear expressions).

```
public int GetIsoWeek(DateTime dt) {
DateTime week1;
int IsoYear = dt.Year;
if (dt >= new DateTime(IsoYear, 12, 29)) {
week1 = GetIsoWeekOne(IsoYear + 1);
if (dt < week1) {
week1 = GetIsoWeekOne(IsoYear);}
else {IsoYear++;}}
else {
week1 = GetIsoWeekOne(IsoYear);
if (dt < week1) {
week1 = GetIsoWeekOne(--IsoYear);}}
return (IsoYear * 100) + ((dt - week1).Days / 7 + 1);}}
```

I am stuck!

- Sometimes refactoring is just not enough.
- Sometimes I just need to rewrite my code.
- Professionals rewrite their code regularly
- Why would my first attempt be any good?

Problem: iso++ and --iso

- There are three cases we need to look at. The date we're given is:
  - less than the date for week one. We calculate the week number based on the previous year.
  - greater than or equal to the date for week one of the following year. We calculate the week number based on the next year.
  - in between those two values for week one. We calculate the week number based on the date's year.

Lets lose week1

```
public  int GetIsoWeek(DateTime dt) {

int IsoYear;
if (dt < GetIsoWeekOne(dt.Year))
IsoYear = dt.Year - 1;
else if (dt >= GetIsoWeekOne(dt.Year + 1))
IsoYear = dt.Year + 1;
Else IsoYear = dt.Year;
return (IsoYear * 100) + ((dt - GetIsoWeekOne(IsoYear)).Days / 7 + 1);}
```

Let's extract that return expression

```
private int CalculateIsoWeek(DateTime date, int isoYear) {
return (isoYear * 100) + ((date - GetIsoWeekOne(isoYear)).Days / 7 + 1);}

public int GetIsoWeek(DateTime dt) {
int IsoYear;
if (dt < GetIsoWeekOne(dt.Year))
IsoYear = dt.Year - 1;
else if (dt >= GetIsoWeekOne(dt.Year + 1))
IsoYear = dt.Year + 1;
else IsoYear = dt.Year;
return CalculateIsoWeek(dt, IsoYear);}
```

Cancelling the final temp

```
public  int GetIsoWeek(DateTime date) {
  if (date < GetIsoWeekOne(date.Year))
      return CalculateIsoWeek(date, date.Year - 1);
  if (date >= GetIsoWeekOne(date.Year + 1))
      return CalculateIsoWeek(date, date.Year + 1);
  return CalculateIsoWeek(date, date.Year);
}
```