

CSC7322: Object Oriented Development

J Paul Gibson, A207

`paul.gibson@it-sudparis.eu`

<http://www-public.it-sudparis.eu/~gibson/Teaching/CSC7322/>

Fundamental Concepts (In Java)

[.../~gibson/Teaching/CSC7322/L1-FundamentalConcepts.pdf](http://www-public.it-sudparis.eu/~gibson/Teaching/CSC7322/L1-FundamentalConcepts.pdf)

Moving from procedures to objects/classes

Questions:

- What is good about the procedural approach?
- What is bad about the procedural approach?
- What could we force the user to do to make the procedural approach better?
- Could we build these constraints into the language?
- What can we help the user to do to avoid repetition?
- Could we build these features as semantic extensions to the language?
- For example, is this why C moved to C++?

From Structures to Objects

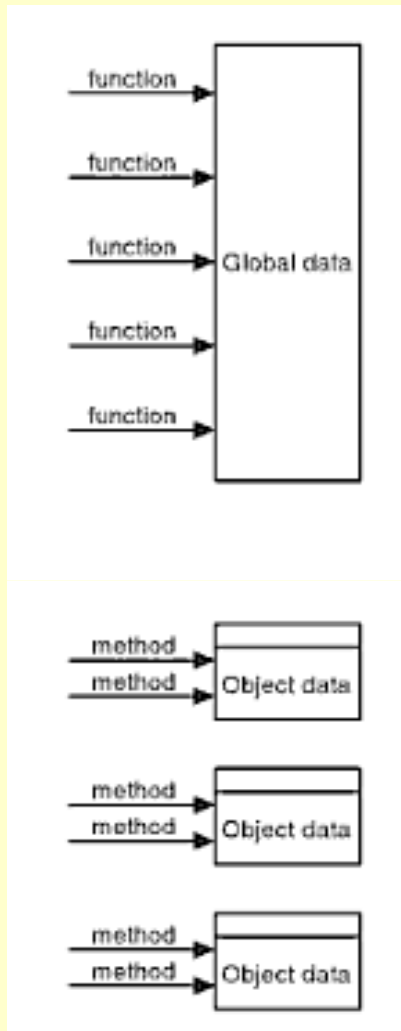
In C and other procedural programming languages:

- programming is action oriented
- unit of programming is the function (or procedure)
- data exists to support the actions that the functions perform
- dynamically, functions are called

In Java and C++ (and other OO languages):

- programming is object oriented
- unit of programming is the class (a user-defined type)
- data and functions/methods are contained within a class
- dynamically, objects are created (and used through their methods)

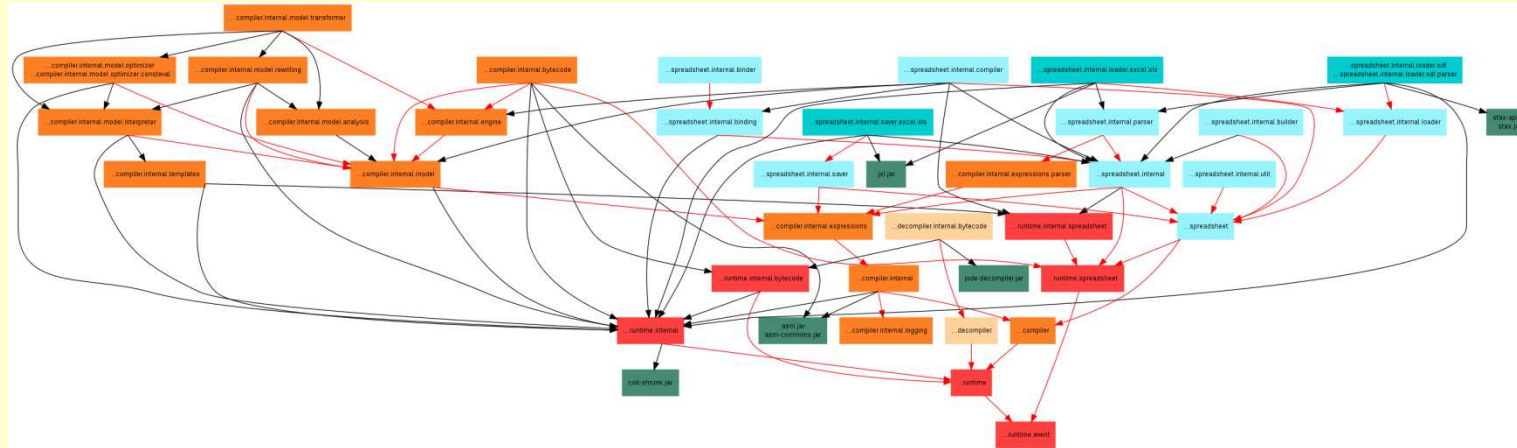
OO is a structuring mechanism



OO is more than just a means of structuring data and functions:

But it does help in structuring our models

Before OO: the function-data problem

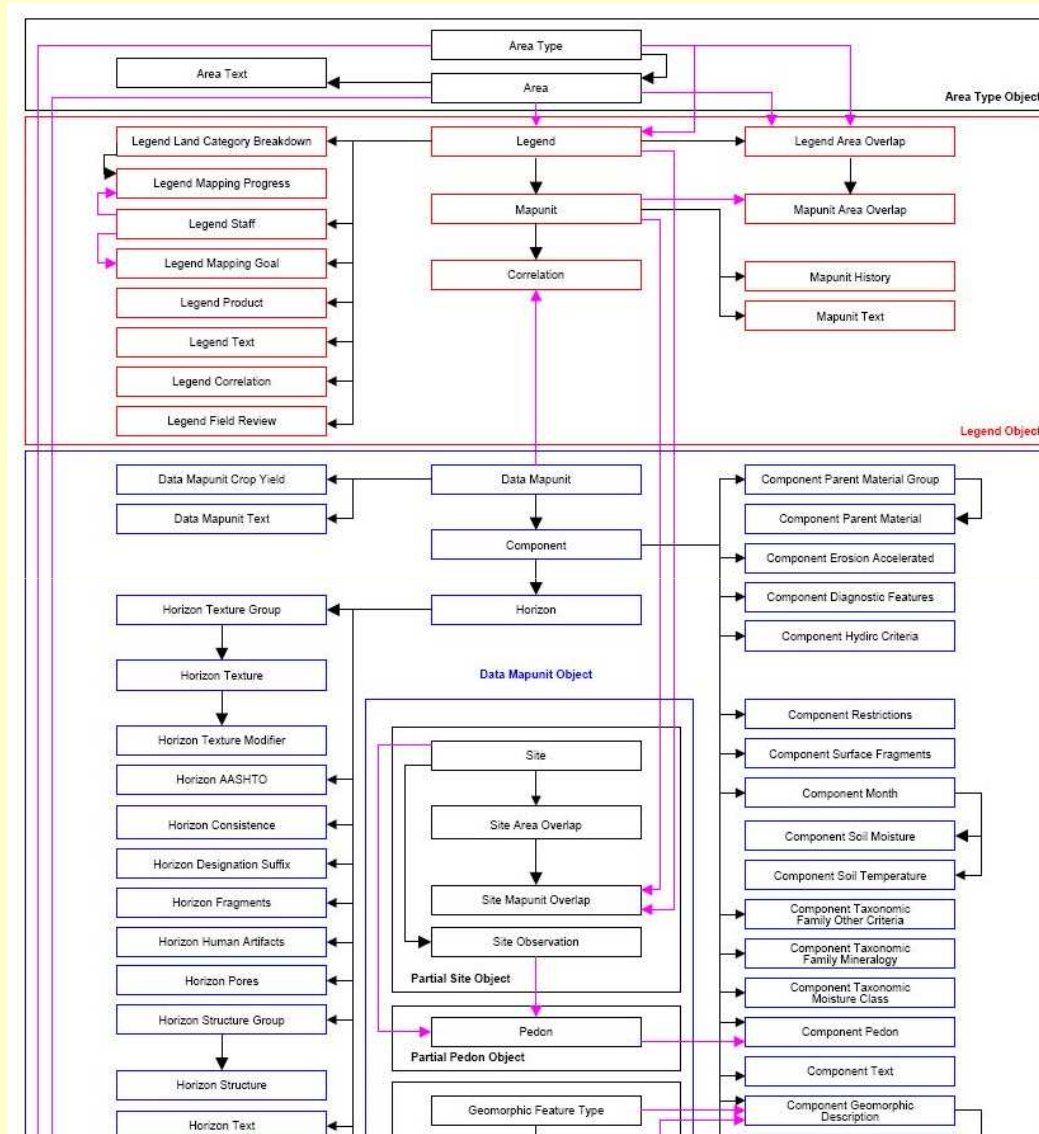


Functional dependencies provided a structural view

Partitioning the dependency graph (coloured above) provides a means of grouping functions into modules/packages

However, the data structure is not so evident (it is moving around the functions and it is stored in modules).

Before OO: the function-data problem



A typical data structure diagram does not help in structuring understanding of the functionality of the system

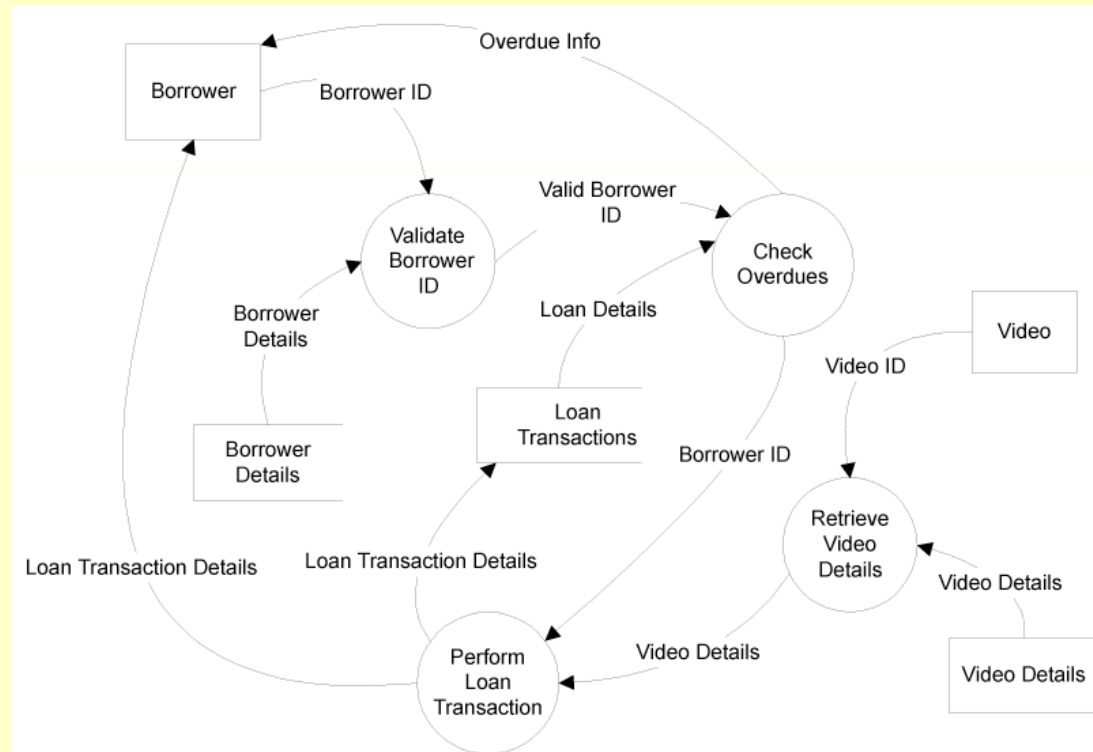
Before OO: the function-data problem

The real problem is integrating dynamic and static views/models/properties/requirements in a coherent fashion (and facilitating different levels of abstraction)

A good example is a DFD

But, this is a compromise which is focussed on what a system does rather than how it does it.

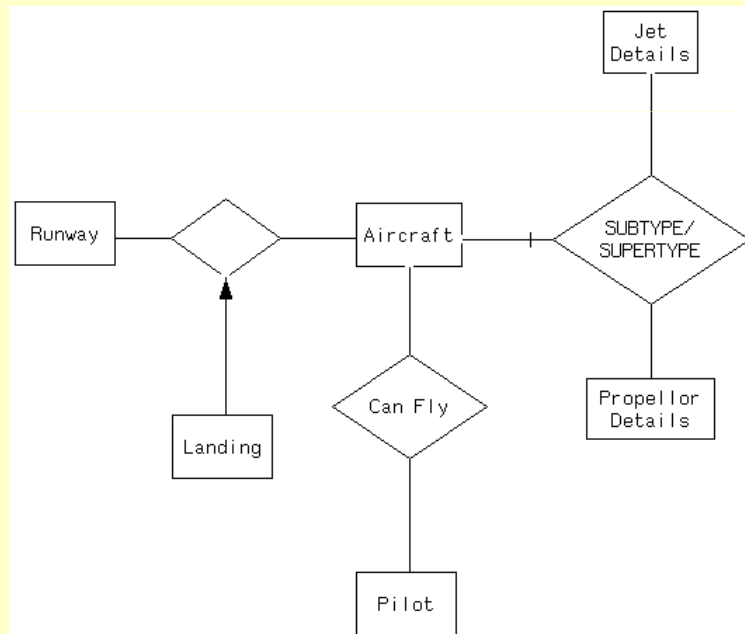
Question: how well does it integrate the dynamic and static views?



Before OO: the function-data problem

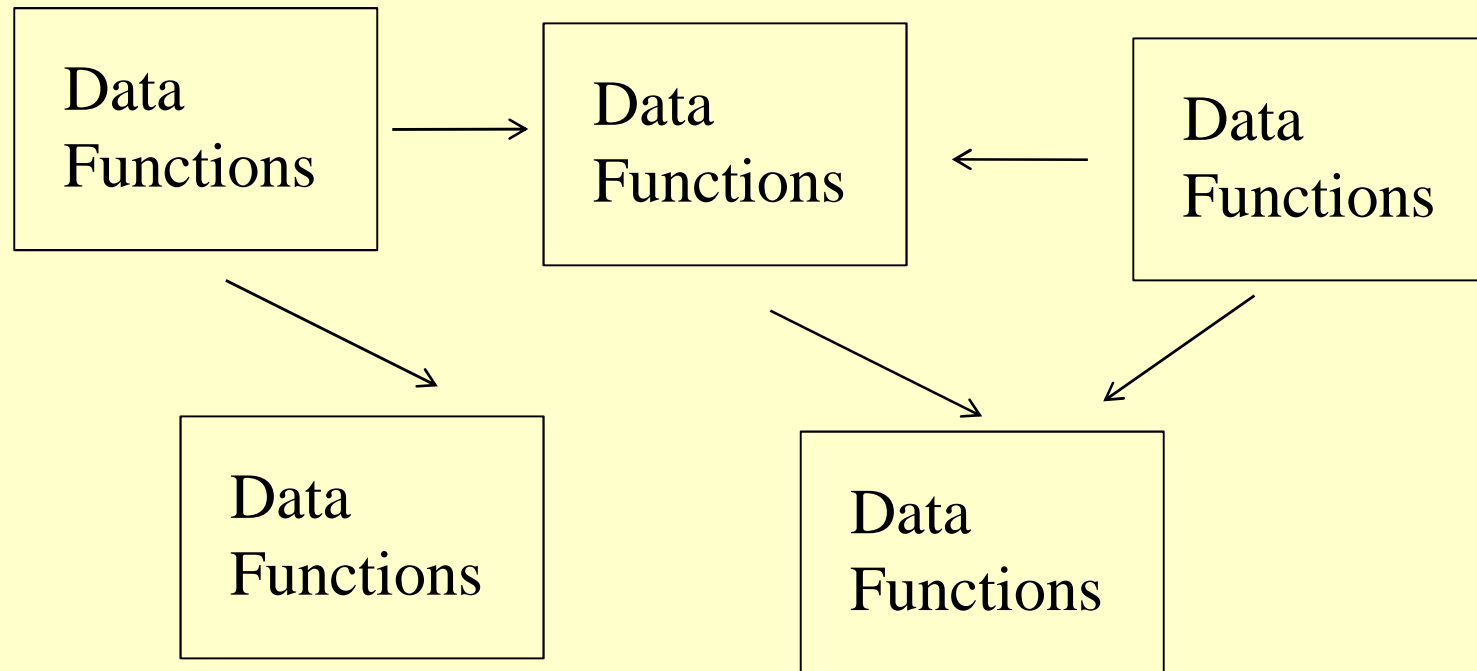
The real problem is integrating dynamic and static views/models/properties/requirements in a coherent fashion (and facilitating different levels of abstraction)

Another good example is entity relationship diagrams



Question: would you say these focus on dynamic behaviour or static structure?

After OO: the function-data problem



Constructing Software Systems

Given a system which is structured in terms of functions and data, one must ask how to construct the system.

Further, one must ask whether the decomposition mechanism supports re-use.

Also, does it help in system evolution/maintenance

Finally, one should ask about support for independent development/compilation

QUESTION: Do you know what the major issues were before OO and whether OO has helped/hindered/changed the situation?

Constructors: make an object of a class ... make a system

Constructors are fundamental to all OO programming:

- A special method/function used to initialise data members in the object being created
- Called automatically when an object of the class is created
- Common to have several constructors for a class (overloading)
- Things get complicated when:
 - member objects of a class are themselves objects!
 - we inherit behaviour ...
 - we have recursive data structures in classes

There are also **destructors**.

Question: why are these not so fundamental in a language like Java?

Composition

This is also called containment or aggregation

When a class/object refers to the use of another class/object

Lots of types of composition ... are there useful categories?

Composition is the most common re-use mechanism in OO

Other re-use mechanisms are [genericity](#) and [inheritance](#).

INHERITANCE

Another way to reuse existing software

Also called specialisation or derivation or extension

Has a lot of other consequences (polymorphism and dynamic binding)
which we will come to later

Often referred to as the 'is-a' relationship

It is different in every single OO language I have ever seen!

In Java it is not particularly complicated

We will try to understand it in isolation before we address the complexities

Some 'simple' examples will help...

Parent constructors and destructors

When an object of a derived class (a subclass) is declared, then the parent's **default** constructor is called before the required constructor of the derived class

If the parent has a parent then the grandparent's **default** is called followed by the parent's **default** followed by the derived class constructor

In effect, for an arbitrary number of parents, the parent **default** constructors are called in a top-down fashion.

The same thing happens with the destructors except it is down bottom-up (with the derived child class calling its own destructor first)

NOTE: This can get very complicated (esp. with multiple inheritance, which is not a problem in Java)

Dynamic binding and polymorphism

Dynamic binding:

when the class of the object is used (at run time) to decide which method gets executed (its own method)

So, for example, if we define a Pets class with 2 subclasses (dog and cat). Each offers a method makenoise(). If we declare an object, Pugsy say, to be a Pet, then Pugsy may bark or purr depending on **whether it is a dog or a cat**.

The key is that we cannot tell at compile time whether Pugsy is a dog or a cat so the correct method can only be decided at run time

Unless we want the Pet method makenoise every time (which we usually do not) then we need **some mechanism to tell the compiler to bind the method at run time**.

This mechanism is dynamic binding

Abstraction

Abstraction is the most important part of any modelling activity

When working with software models there are standard abstractions:

- Control abstraction
- Data abstraction
- Class - an attempt to combine abstractions of data and code.

Computer science commonly presents levels/layers of abstraction, wherein each level represents a different view/model of the same “thing”.

QUESTION: in OO we talk about encapsulation, but how is this related to abstraction?

Dice Class Example: to illustrate fundamental concepts

We wish to be able to construct a virtual/electronic dice.

The dice will have by default six sides (numbered 1..6)

We can, if we wish, also construct a dice with any number of sides between a minimum of 3 and a maximum of 36. We can also construct a new dice as a copy of an existing dice.

The dice provides a main service/function/method – you can roll it and see the result of this last roll. The last roll value is initially set to 0 before the dice is rolled.

Each dice also provides a secondary service – it counts the number of times it has been rolled since it was created, and allows this value to be read.

We must be able to make many instances of the Dice. Our system will count the number of instances that are currently in existence.

Finally, our Dice must implement standard methods: invariant, to String and equals

Dice Class Example: to illustrate fundamental concepts

models.Dice

The original natural language specification -

- *We wish to be able to construct a virtual/electronic dice.*
- *The dice will have by default six sides (numbered 1..6)*
- *We can, if we wish, also construct a dice with any number of sides between a minimum of 3 and a maximum of 36. We can also construct a new dice as a copy of an existing dice.*
- *The dice provides a main service/function/method – you can roll it and see the result of this last roll. The last roll value is initially set to 0 before the dice is rolled.*
- *Each dice also provides a secondary service – it counts the number of times it has been rolled since it was created, and allows this value to be read.*
- *We must be able to make many instances of the Dice. Our system will count the number of instances that are currently in existence.*
- *Finally, our Dice must implement standard methods: invariant, to String and equals*

Version:

1

Author:

J Paul Gibson

Dice Class Example: to illustrate fundamental concepts

As well as the Dice class – in a models package, you are to write:

- A Unit test class (with JUnit)
- A simulation test class (using RNG)
- Full documentation of model and test code

You are also to consider future extensions to:

- the model
- UI

Extending the Dice class with statistics

DiceWithStatistics

This class provides the behaviour of a Dice of any positive number of sides
The values on each side are integers ranging from 1 up to the number of sides
Statistics for the frequencies of roll values is stored
These statistics are readable through the toString method

Version:

1

Author:

J Paul Gibson

We will use the Java inheritance mechanism so that we define this new class as a subclass (extension) of the Dice class

If you do not manage to develop the first Dice class then you should use my sample solution

My Dice Sample Solution

models

- Dice

tests

- JUnit_DiceTest
- Random_DiceTest

tools

- DateHeader
- HasInvariant
- InvariantBroken
- SeedRNGCommandLine

You should download these from the website:

<http://www-public.int-evry.fr/~gibson/Teaching/CSC7322/Code/Dice.zip>

TO DO: Create a new class which extends DiceWithStatistics

This class should add a new method that will ‘draw the dice frequencies’ as a histogram in the console

The frequencies can either be displayed **vertically** or **horizontally**. If the frequency values are bigger than the height/width of the screen then you should scale them in order to occupy as much of the screen as possible. The screen size should be stored in constant variables HEIGHT and WIDTH.

Add a new boolean attribute for the type of display and another method to change this value.

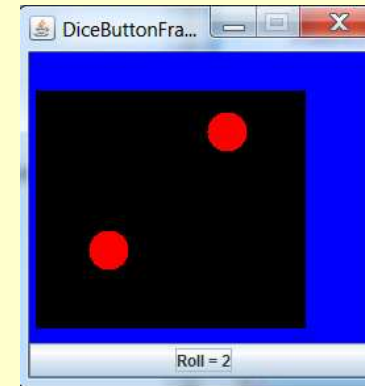
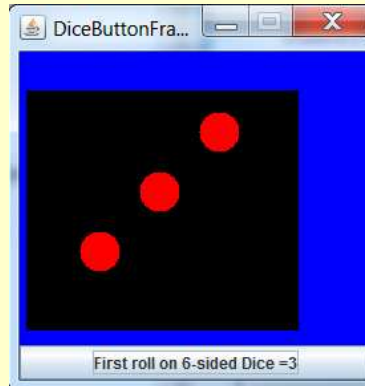
```

                *
            *   *       *
        *   *   *   *
    *   *   *   *   *
* * * _ * * *
                |***
                |*
                |****
                |
                |*
                |**
                |***

```

TO DO LATER: Create a new class which extends the 2 Dice classes each with a GUI (using Swing/AWT)

Example Dice GUI



TO DO LATER :
Design and code a
DiceWithStatistics GUI

My Dice Solution – check out the documentation that can be generated using Javadoc comments

tests.JUnit_DiceTest

Test [Dice](#) Using JUnit

- test the default constructor [Dice.Dice](#)
- test the non default constructor [Dice.Dice\(int\)](#)
- test the copy constructor [Dice.Dice\(Dice\)](#)
- test exception in copy constructor [Dice.Dice\(Dice\)](#)
- test [Dice.roll](#)
- test exception in [Dice.lastRoll](#)
- test exception in [Dice.setRNG\(java.util.Random\)](#)
- test [Dice.Equals](#)
- test [Dice.toString](#)

Version:

1

Author:

J Paul Gibson

See Also:

[Random_DiceTest](#)

My Dice Solution

tests.Random_DiceTest

Test class for [Dice](#) that uses a [Random](#) RNG for simulation purposes.
The RNG can be seeded at the command line, or a default value of 0 can be used.

We use the [DateHeader](#) class to document the date/time of the test execution

Expected Output (using default RNG seed = 0) and NUMBER_OF_TEST_ROLLS = 6:

```
The seed used for the random number generator in the test is 0.  
You can override this value by passing an integer value as a main argument parameter, if you so wish.
```

```
*****  
Execution Date/Time 2011/01/31 16:38:43  
*****
```

```
6-sided Dice - lastRoll = 0. (Total number of rolls = 0)  
Rolling 6 times  
1. Roll = 1  
2. Roll = 5  
3. Roll = 2  
4. Roll = 6  
5. Roll = 6  
6. Roll = 6  
6-sided Dice - lastRoll = 6. (Total number of rolls = 6)
```

Version:

1

Author:

J Paul Gibson

My Dice Solution

`models.Dice.Dice()` throws `InvariantBroken`

Dice constructor by default will make a dice with `DEFAULT_numberOfSides` sides.
Increments the `numberOfDie`

Throws:

[InvariantBroken](#) - if Dice was not constructed in a safe state

My Dice Solution

`models.Dice.Dice(int sides)` throws `InvariantBroken`

Dice constructor with the specified number of sides

Increments the `numberOfDie`

If the specified number `sides` is bigger than `MAXIMUM_numberOfSides` or less than `MINIMUM_numberOfSides` then the default of `DEFAULT_numberOfSides` is used.

Parameters:

sides defines the number of sides for the new dice being constructed.

Throws:

[InvariantBroken](#) - if Dice was not constructed in a safe state

My Dice Solution

🌱 **models.Dice.Dice(Dice diceToCopy) throws IllegalArgumentException, InvariantBroken**

Dice copy constructor - which copies the `numberOfSides` and the `lastRoll` values, it does not copy the `numberOfRolls` value as this is initialised to 0 in the new Dice.

Increments the `numberOfDie` .

Parameters:

diceToCopy provides the values to be copied in the new Dice.

Throws:

[InvariantBroken](#) - if Dice was not constructed in a safe state

[IllegalArgumentException](#) - if argument `diceToCopy` is null

My Dice Solution

- **void models.Dice.roll() throws InvariantBroken**

Rolls the dice and updates the last roll to the random value rolled, and increments the roll count
The design decision was to separate the roll from the reading of the value

Throws:

[InvariantBroken](#)

My Dice Solution

- **void models.Dice.setRNG(Random rng) throws IllegalArgumentException, InvariantBroken**

Reset the random number generator

Parameters:

rng is the new random number generator to be used when rolling the dice

Throws:

[InvariantBroken](#) - if Dice was not constructed in a safe state

[IllegalArgumentException](#) - if argument `rng` is null

My Dice Solution

- `boolean models.Dice.equals(Object thing)`

Overrides: [equals\(...\)](#) in [Object](#)

Parameters:

thing is the Object to test for equality

Returns:

true if

- the two objects reference the same address, or
- the object is a Dice and there is equality on `lastRoll` and `numberOfSides` fields

otherwise false

My Dice Solution

- **String models.Dice.toString()**

Overrides: [toString\(\)](#) in [Object](#)

Returns:

The current state of the Dice as a String

The format to be followed is:

```
6-sided Dice - lastRoll = 0. (Total number of rolls = 0)
```

My Dice Solution – Fix a Problem/Bug

In my JavaDoc documentation,
of Dice and its tests, there is a small bug

Can you find it and fix it?

A Preview of things to come in the next lecture:

Choosing to implement a `DiceWithStatistics` as a subclass of `Dice` is a design decision that could have consequences on later stages of the development.

An alternative re-use mechanism is to design a `DiceWithStatistics` class/object to have a `Dice` component class/object.

If you wish, attempt to implement this second design.

In the next PBL session we will see the consequences of each of these designs when we try to implement an observable `Dice` that is observed by a `Dice View`.