

Week 8, Lecture 1

Introducing Inheritance



1

Announcements

- The deadline for Lab Assessments 4 and 5 is **this week**.
- Assignment 2 has been marked. Your mark and commented code is on the web page.

2

This Week

Lecture 1: Introducing inheritance

Lecture 2: Case study: a day at the zoo

Java Genesis:

–Ch8: Inheritance

Lab Assessment 6 (deadline Week 10)

Quick Quiz for Chapter 7

3

Object-oriented Reuse

1. Instantiation:

can create objects of a class

2. Inheritance:

can create a new class that is 'like'
an existing class

4

Example

1. Instantiation:

to cook a chocolate cake, reuse the
chocolate cake recipe

2. Inheritance:

to create a recipe for cherry cake, take
the chocolate cake recipe and replace
'chocolate' by 'cherries'

5

Another Example

1. Instantiation:

new cars are produced by the car factory

2. Inheritance:

to build a new car factory, reuse the
plans for the existing factory and
modify them as required

6

Extending CircleFigure

Extend the class **CircleFigure** to include a method

```
public static void smoothRight (int x)
```

which when invoked moves the circle right **x** pixels at the rate of 1 pixel every 10 milliseconds.

7

CircleFigure

```
moveRight (int)
moveLeft (int)
moveUp (int)
moveDown (int)
moveTo (int, int)
setRadius (int)
setColour (Color)
drawHollow ()
drawFilled ()
getXCentre () →int
getYCentre () →int
```

8

CircleFigure

```
moveRight (int)
moveLeft (int)
moveUp (int)
moveDown (int)
moveTo (int, int)
setRadius (int)
setColour (Color)
drawHollow ()
drawFilled ()
getXCentre () →int
getYCentre () →int
```

BetterCircleFigure

```
smoothRight (int)
```

9

```
import genesis.*;

public class BetterCircleFigure
    extends CircleFigure {

    public static void smoothRight (int x) {
        int dist = Math.abs(x);
        for (int i=1; i<=dist; i++) {
            Delay.milliseconds(10);
            if (x>0) moveRight(1);
            else moveRight(-1);
        }
    }
}
```

10

Tossing Dice

An object of the class **DiceGame** denotes:

- a game consisting of a number of dice;
- each die has a current value (in the range 1 to 6);
- the dice can be tossed with the result a new current value for each die is randomly chosen;
- the sum of the current values of the dice can be calculated;
- the current value of each die and the sum of these values can be output.

11

```
public class DiceGame {

    private int number; // instance variables
    private int [ ] currentValues;

    public DiceGame (int num) { // constructor
        number = num;
        currentValues = new int [num];
    }

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

    public void tossAll () {
        for (int i=0; i<number; i++) {
            currentValues[i] = randInt();
        }
    }
}
```

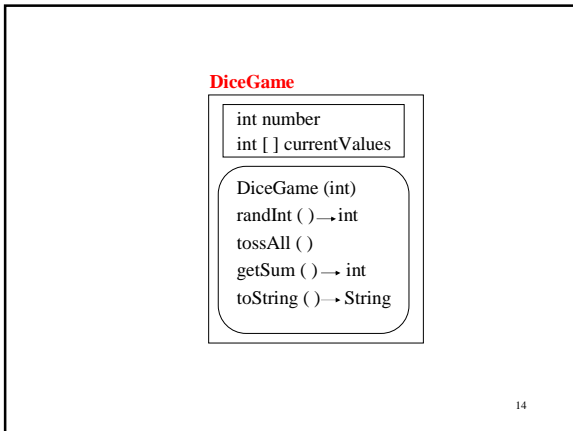
12

```

public int getSum ( ) {
    int sum = 0;
    for (int i=0; i<number; i++) {
        sum = sum + currentValues[i];
    }
    return sum;
}

public String toString ( ) {
    String data = "";
    for (int i=0; i<number; i++) {
        data = data + currentValues[i] + " ";
    }
    data = data + " " +getSum();
    return data;
}
}

```



Tossing Coins

Observation:

Tossing coins is much like tossing dice except the outcome of each toss is 0 or 1 (for heads and tails respectively).

Therefore to create coin tossing games construct a class **CoinsGame** that inherits (extends) the class **DiceGame** and modifies the code that determines the outcome of a toss.

```

public class CoinsGame extends DiceGame {
    public CoinsGame (int num) {
        super(num);
    }
    public int randInt ( ) {
        return super.randInt() % 2;
    }
}

```

