

# 'Objectsville'

covering

\*\* Procedural *versus* Object-Oriented programming



Chapters 2-4 – “Head First Java” book

Chapters 4+6 – “Introduction to Java Programming” book

# Concepts: OO programming and objects

## What is OO programming?

- Constructing software systems which are **structured collections** (or sets) **of classes**.
- These classes produce instances called **objects**, which **communicate with each other using messages**.

Imagine that you are building a database about cars; you would have a Car object.

## What is an object?

- An **object** is a thing; it is a fundamental entity in Java.
- Objects tend to be the **nouns** in specifications.
- In software terms, e.g.
  - car;
  - bank account;
  - student;
  - employee;
  - complex number;
  - GUI button.

**OO** = Object-Oriented

# What is not an object?

- **Attributes** (or **states**) of an object: essentially anything that **describes or quantifies** an object.
  - speed, colour, make, model, and position are all attributes of a **car object**;
  - number, owner, balance might be attributes of a **bank account object**.
- **Operations** (or **behaviours**) of an object: they mostly correspond to **verbs** in a requirements specification.
  - turn left, speed up, slow down, turn right are all operations of a **car object**.
  - open, close, deposit, withdraw, are all operations on a **bank account object**.

# What is a class?

- An **object** is **defined** by a **class**.
- The **class** defines the **attributes** and **operations** exposed by one or more related objects
- In Java, a **class** is to:
  - define a kind of object or in other words, to **define a data type**



## Classes *versus* Objects

An **object** is an **instance** of a particular **class**.

- In the examples so far, all the code went in the **main()** method.



This is **not** object-oriented!

- How does object-oriented programming change how we do things?
- We **can split up code between different objects** ...



... and things for you to try out!

# Real world objects (1/3)

## Person

To **describe a person**: name, gender, age, occupation, ...

A **person can do**: eat, drink, sleep, walk, ...

object

object



Jane  
female  
19  
Student  
...

Emma  
female  
45  
Doctor  
...

## Car

To **describe a car**: make, model, year, colour, ...

A **car can do**: accelerate, brake, turn, reverse, ...

object

object



BMW  
M3  
blue  
...  
...



Mini  
Cooper  
red  
...  
...

# Real world objects (2/3)

What are the **attributes** and **operations** of a mobile phone?

Mobile phone

Attributes:

Operations:

object



iPhone  
5  
Vodafone

...

object



Samsung  
Galaxy S3  
Orange

...

object



BlackBerry  
Curve 8900  
O2

...

# Real world objects (3/3)

## Bank account

### Attributes

account number  
account name  
account type  
sort code  
address  
balance  
overdraft limit  
...

### Operations

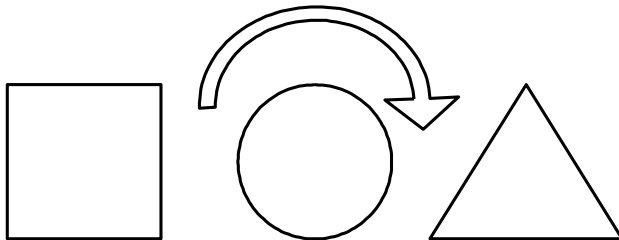
view details  
print statements  
check balance  
deposit  
withdraw  
change address  
change overdraft limit  
...



# OO *versus* Procedural Programming

## The Specification

There will be shapes on a GUI: a square, a circle and a triangle. When the user clicks on a shape, the shape will rotate clockwise 360° (i.e. all the way around) and play an MP3 sound file specific to that particular shape.



- The **task**: to create a program that fulfills the following specification
  - Procedural approach
    - What does the program have to do?
    - What procedures are needed?
      - **rotate** and **playSound**

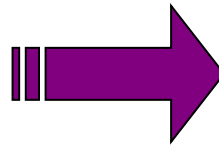


## Procedural

```
rotate (shapeNum) {  
    // make the shape rotate 360°  
}  
  
playSound (shapeNum) {  
    // use shapeNum to loop-up which  
    // sound to play and play it  
}
```

# OO: questions to answer (1/2)

- What are the **things in this program**, i.e. what are the **objects**?
  - Objects are **things or nouns**.
  - The main interacting force of this program is of course the **shapes**.



## The Specification

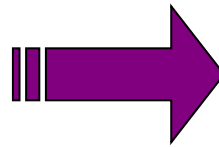
There will be **shapes** on a **GUI**, a **square**, a **circle** and a **triangle**. When the **user** clicks on a **shape**, the **shape** will rotate clockwise 360° (i.e. all the way around) and play an MP3 **sound file** specific to that particular **shape**.



There are other **objects**, but this is enough to start with.

# OO: questions to answer (2/2)

- **Verbs** indicate the **actions** of an **object**.



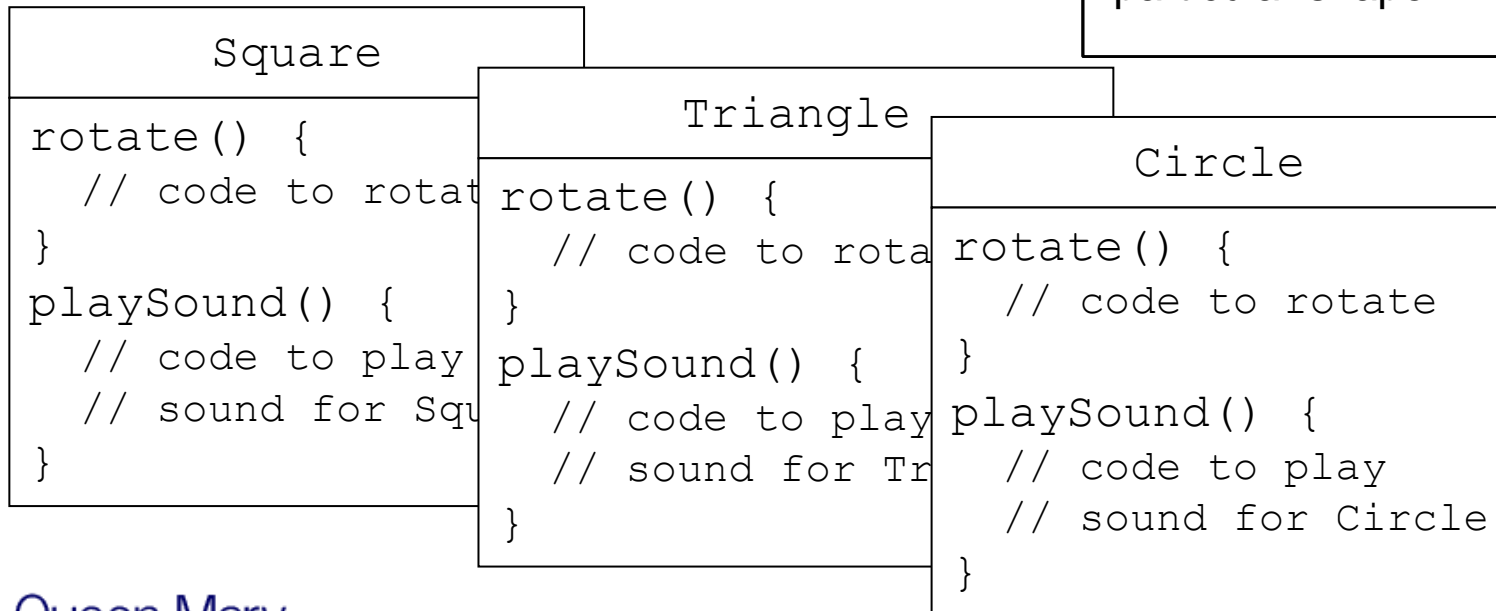
## The **Specification**

There will be shapes on a GUI, a square, a circle and a triangle. When the user **clicks** on a shape, the shape will **rotate** clockwise 360° (i.e. all the way around) and **play** an MP3 sound file specific to that particular shape.



In Java, objects correspond to classes!

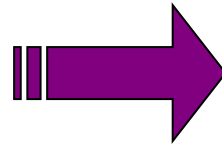
## Object-Oriented



# What if the specification changes ...

## Addition to Specification

A fourth shape is needed – a random shape. When the user clicks on the random shape, it will rotate and play a WAV sound file.



## Procedural

```
playSound(shapeNum) {  
    // if shape is not a random shape  
    // as before..  
    // else  
    // play random shape  
}
```



Always try to minimise altering code you have already tested.  
**Changes could introduce errors!**

- `rotate()` will still work, as the code uses a lookup table to match a shape to a graphic.
- However, `playSound()` has to change!

## Object-Oriented

```
RandomShape  
  
rotate() {  
    // code to rotate  
}  
  
playSound() {  
    // code to play  
    // sound for RandomShape  
}
```



With OO, we do not need to 'touch' any of the code we have already written!

# Procedural and OO programming: again

- In order to account for variation in rotation points, we need to **add some new arguments in the procedural method**.

## Procedural

```
rotate(shapeNum, xPt, yPt) {  
    // if shape is not a random shape  
    // calculate the centre point  
    // based on a rectangle and rotate;  
    // else use passed in xPt and yPt  
    // as rotation offset and rotate  
}
```



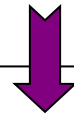
Lots of code has been affected! It will **ALL** have to be recompiled and tested.

new  
attributes



## Object-Oriented

RandomShape
<pre>int xPoint int yPoint rotate() {     // code to rotate } playSound() {     // code to play     // sound for RandomShape }</pre>



New rules for rotation are simply put in the **RandomShape's rotate()** method. No other shape is affected! **The old shapes don't need to be tested again**. In fact, the compiled code for **Circle**, **Square** and **Triangle** doesn't change at all.

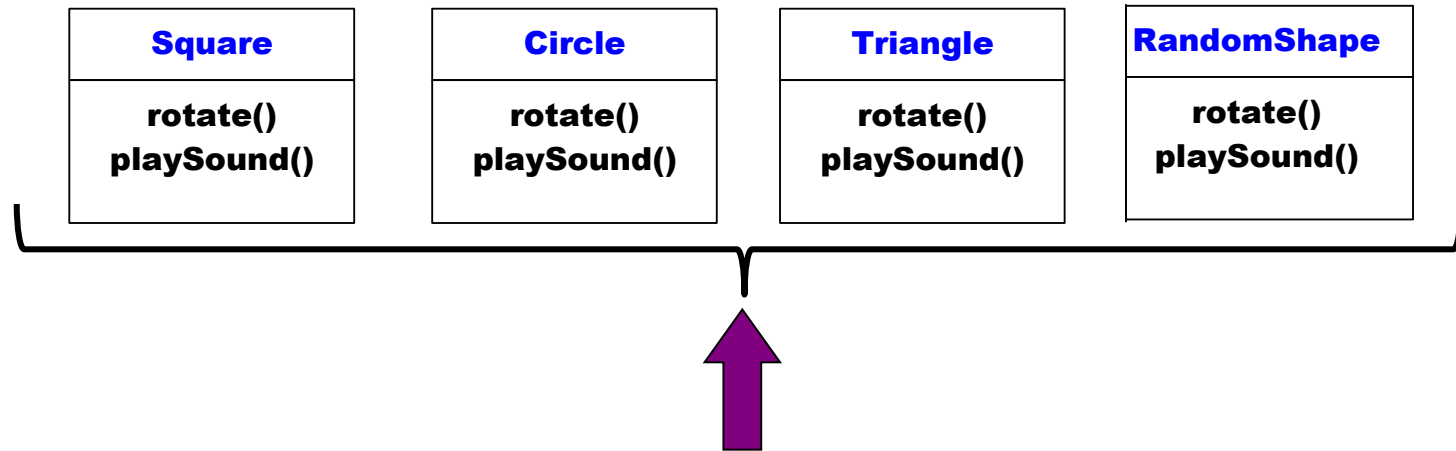
# Summary: Procedural *versus* OO Approaches

- Procedural approach
  - 2 procedures that behave differently based on the shape.
  - If anything about the specification changes, then these 2 methods have to change.
- OO approach
  - 4 classes, with 2 methods each!
    - Many more methods than with the procedural approach!
    - Duplicated code? ← Is this a good thing?
  - However, each object controls its own behaviour.



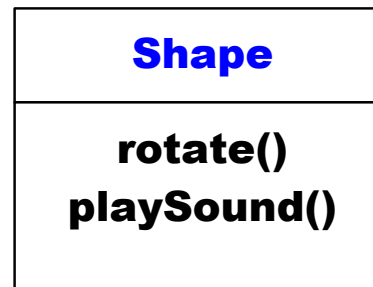
... and things for you to try out!

# Introducing Abstraction and Inheritance



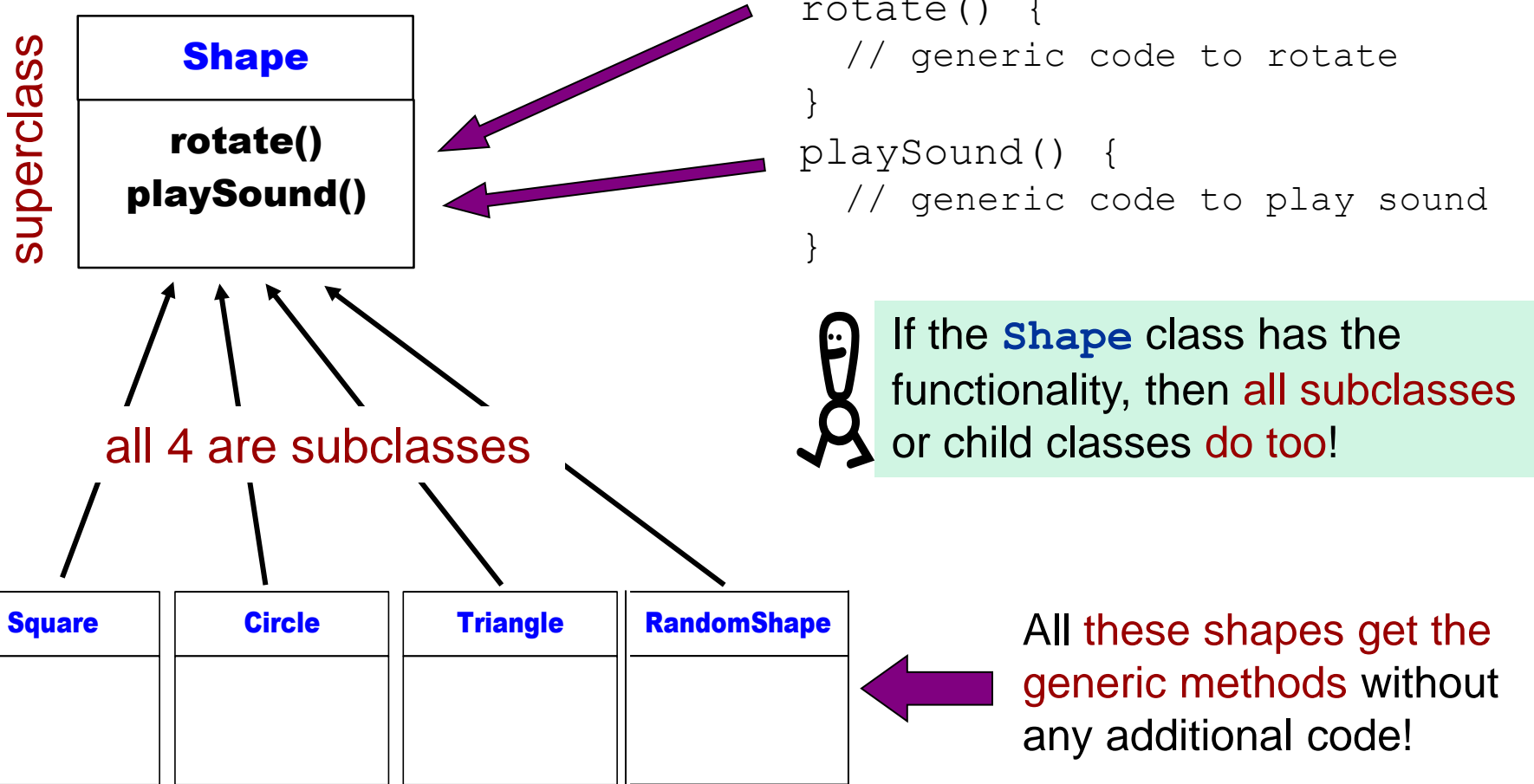
Look at **what they have in common!**

**Abstract** out the  
common features





# Inheritance



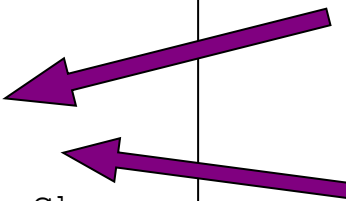
“**Square inherits from Shape**”

# Specialising

- In the case of the **RandomShape**, we provide our own “random shape” specialisations!

RandomShape
<pre>int xPoint int yPoint rotate() {     // code to rotate } playSound() {     // code to play     // sound for RandomShape }</pre>

**Method Overriding** – we  
redefine the inherited methods!  
Every object has its own  
behaviour!





... and things for you to try out!