Programming Techniques

Inheritance and other OO concepts; Static

RMIT UNIVERSITY

Pre-Lecture Videos

- Inheritance
  - https://goo.gl/pPosCW (03:35)
  - https://goo.gl/PgA1tG (02:32)
- Inheritance and polymorphism
  - https://goo.gl/qHZou8 (5:17)
- Extending classes and overriding methods
  - https://goo.gl/4JwjD7 (5:17)
- Overloading methods with different signatures
  - https://goo.gl/8i985T (04:51)
  - https://goo.gl/HdQKZT (03:54)

RMIT UNIVERSITY

Content:

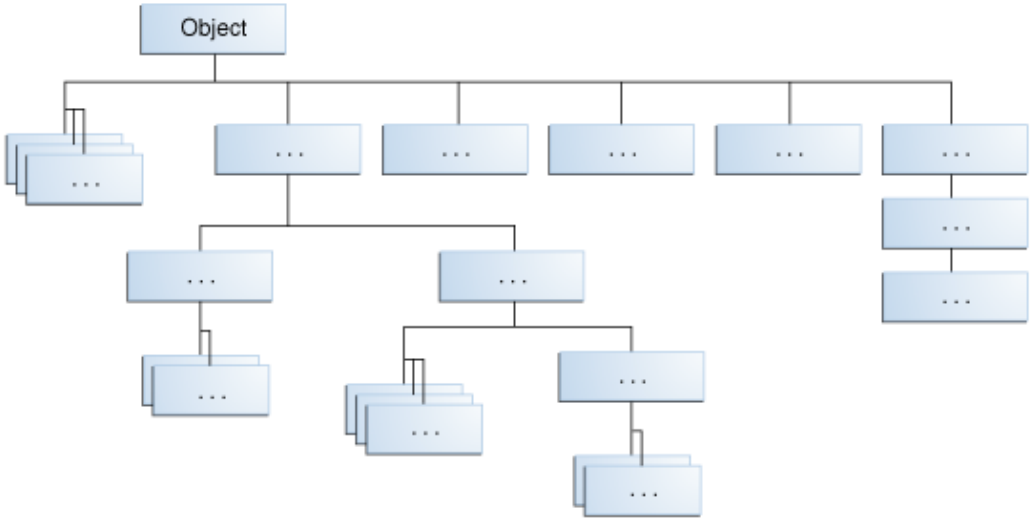
- The concept of inheritance. Java as a hierarchical tree of classes with the Object Class as the parent of all classes.
- Why and how inheritance can be advantageous in your OO programs
- How to design and implement parent and child classes
- The scope of variables, methods and constructors when in an inherited relationship
- Method overloading
- Method overriding
- The super and this reserved words

RMIT UNIVERSITY

Java inheritance - an overview

Classes in an object orientated programming language such as Java are organised into a hierarchy

The **Object** class, defined in the **java.lang** package, defines and implements behaviour common to all classes—including the ones that you write.



https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

Java inheritance - Object class

One of the methods of the **Object** class you have previously used is **equals( )**

```
if(string1.equals(string2))
{
        return true;
}
```

This is just a simple example of when using Java methods that have been inherited from classes higher up the class hierarchy

RMIT UNIVERSITY

https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

Inheritance

Java classes are structured as a tree of related, inherited classes but what does this mean for you?

Inheritance allows you to define a general class and then later define more specialised classes that add some new details to the existing general class definition.

This saves work, because the more specialised class inherits all the properties of the general class, and the application programmer need only program the new features.

RMIT UNIVERSITY

Your own classes and Inheritance

The idea of inheritance is simple but powerful:

When you want to create a new class and there is already a class that includes some of the code that you want, you can derive your new class from the existing class

In doing this, you can reuse the fields (attributes) and methods of the existing class without having to write (and debug!) them yourself.

This means that, in principle, writing code is faster and it is less prone to errors/bugs

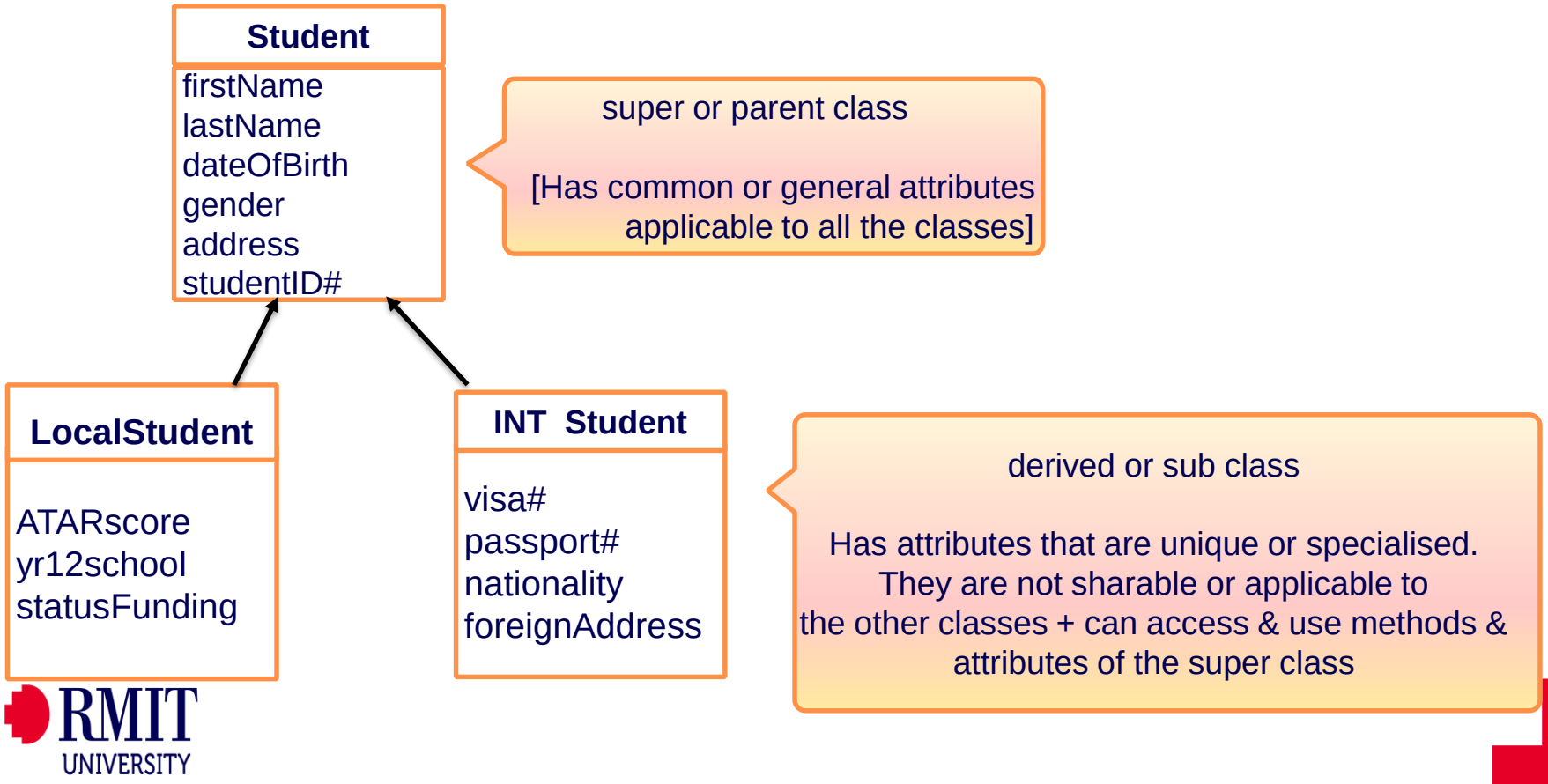
When you are creating classes to model the real world there will be often classes that lend themselves to being inherited

The exact design of the inheritance relationships will depend on the application context and it is something you will learn with practice and experience

https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

super and sub classes

A class that is derived from another class is called a subclass (also a derived class, extended class, or child class). The class from which the subclass is derived is called a superclass (also a base class or a parent class).



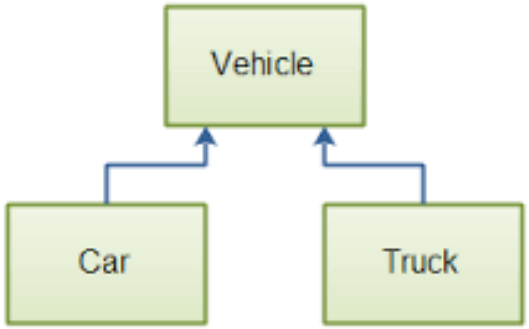
RMIT UNIVERSITY

Question/practice time!

List some attributes that might belong to the **Vehicle** class. *Remember the Vehicle is the parent or super class. It has attributes common to the sub or derived classes* ***Car*** *&* ***Truck***

List some attributes that might be unique to a Car class

List some attributes that might be unique to a Truck class



RMIT UNIVERSITY

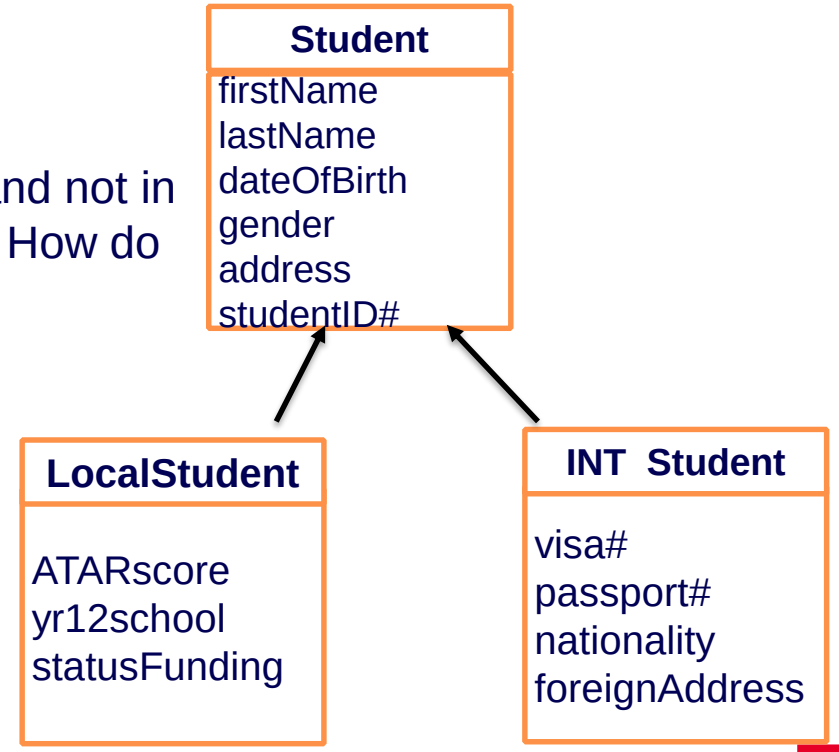
Implementing inheritance

So how do we implement inheritance in our Java code? First of all what are some of the issues?

How do we code the relationship between two classes? What code states the inheritance relationship?

If the dateOfBirth, gender of an **INT Student** & a **LocalStudent** is an attribute in the parent class and not in its own class, how do we represent this in code? How do we assign these values to the parent class

When we instantiate objects of a subclass do we also have to create an object of the parent class?



RMIT UNIVERSITY

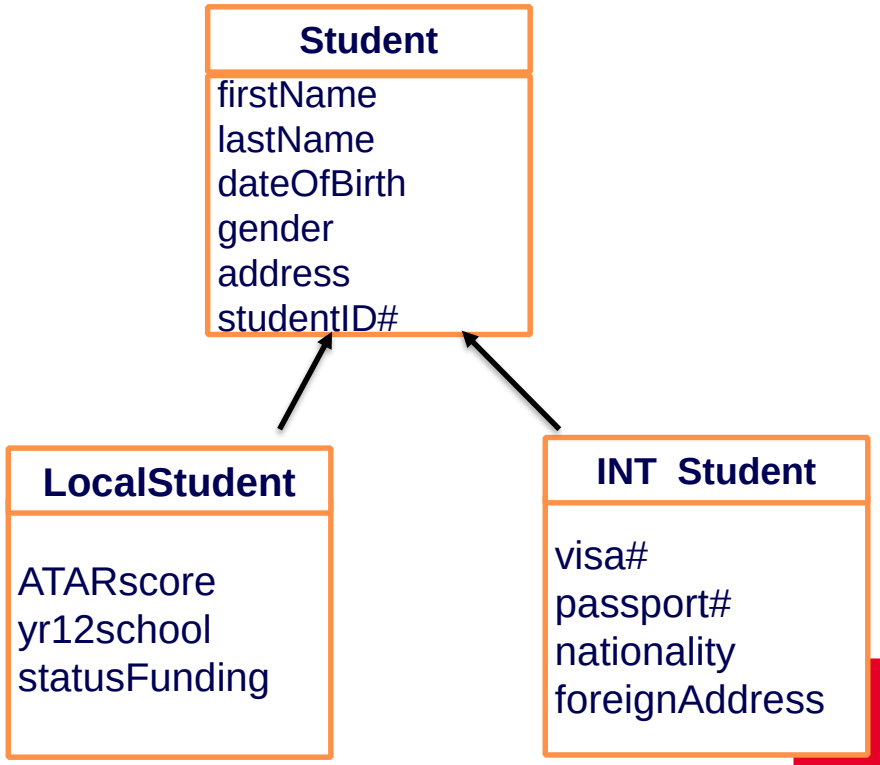
Coding the inherited relationship between classes

```
class Student
{
  ......
}
```

Any child class uses…extends… in its declaration

```
class LocalStudent extends Student
{
 .......   //extends the Student class.
}
```

```
class INTStudent extends Student
{
 .......   //extends the Student class.
}
```



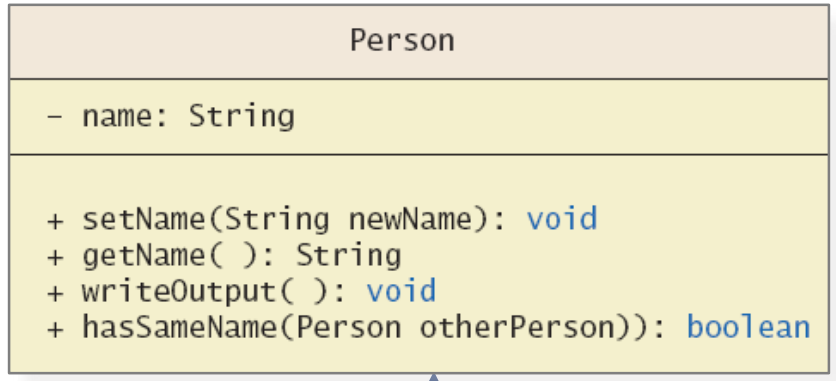
RMIT UNIVERSITY

What can a child object access? Figure 8.3 from Savitch text.

Student class is derived from the **Person** class. Alternatively, we can say the Student class extends the Person class.

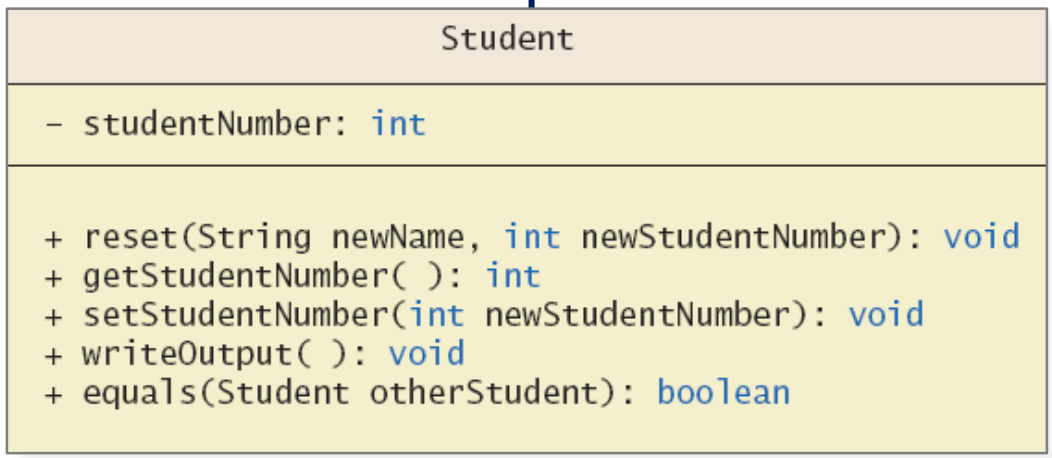
A **Student** object can't (directly) access any private attributes. Can call setter methods

A **Student** object can access - inherits all of the public methods of the parent



Student has own private attribute

Student has own public methods



RMIT UNIVERSITY

Scope of an object of the derived class

An object of the child/sub/derived class can access all public methods in the inherited relationship.

```
Student s = new Student( );    //create a Student object
```

Access to parent private attributes via setter methods. Pass the value to the setter method.

```
s.setName("Buffy")          //call a method in the parent class
s.getName ()                //call a method in the parent class
```

```
s.setStudentNumber( )       //call own method from own class
s.getStudentNumber( )       ///call own method from own class
s.writeOutput()             //call own method from own class
```

RMIT UNIVERSITY

Subclass Construction

- Whenever a subclass object is constructed, the *superclass constructor* must be called.
- Syntax used: keyword super followed by construction parameters if any
- Must be the first statement in the method
- If we omit this statement compiler looks for a *subclass* constructor with no arguments – *default constructor*.

RMIT UNIVERSITY

Subclass constructors - call super class constructors

```
public class Parent
{
        private String name = null;
        private int age = 0;

        public Parent(String name, int age
        {
    this.name = name;
    this.age = age;
        }
```

A constructor for the parent class must be matched with a constructor in the child

```
public class Child extends Parent
{
    private String school = null;

        public Child(String name, int age)
        {
            super(name, age);
        }
```

A matching constructor in the child is used to pass parameters to the parent class in order to assign values to the super class attributes

The reserved word super( ) is a call to the parent constructor. The program moves to the parent class constructor & passes any parameters

RMIT UNIVERSITY

Another example - Using child & parent constructors with inheritance

A child object can set parent class (private) attributes via a setter method but this can also be done via linking parent & child constructors. But how does a child object access parent constructors? The **Person - Student** example shows this.

```
public Person( )
{
    name = "No name yet";
}

public Person(String initialName)
{
    name = initialName;
}
```

Constructor with one argument

```
//instantiate an object using a Student class constructor
Student p = new Student("Gromit",  15901237);
```

Constructor with two arguments

How do you pass the **name** parameter to the **Person** constructor and assign the studentNumber parameter to the **Student** class?

RMIT UNIVERSITY

Using child & parent constructors with inheritance

Instantiating an object of the **Student** class is done in the normal way. If the new operator has arguments then Java will seek the matching constructor.

```
Student s = new Student("Clark Kent", 15903456 );
```

```
public class Student extends Person
{
    private int studentNumber;

    public Student(String initialName, int initialStudentNumber)
    {
        super(initialName);
        studentNumber = initialStudentNumber;
    }
```

NB: The **studentNumber** parameter is an attribute of the child class but **initialName** is an attribute of the parent class. What is super( ) doing?

RMIT UNIVERSITY

Using child & parent constructors with inheritance

The first line of the sub class constructor.. super() is a call to the parent class constructor and passes it the initialName value which is then assigned to the parent attribute name.

```
public class Person
{
   private String name;

   public Person(String initialName)
   {
      name = initialName;
   }
```

The common, parent class attributes are passed to it via the child class constructor calling **super( )**

```
public class Student extends Person
{
   private int studentNumber;

   public Student(String initialName, int initialStudentNumber)
   {
      super(initialName);
      studentNumber = initialStudentNumber;
   }
```

Constructor. One parameter is used by the super( ) constructor and the other assigned to the attribute of this class

Using …. super … to refer to the parent class

- **A Constructor in a subclass must invoke a constructor from the base / parent class using the reserved word `super`**

- **It must be the first statement in the child constructor**

```
public Student(String initialName, int initialStudentNumber)
{
    super(initialName);
    studentNumber = initialStudentNumber;
}
```

The super reserved word can also be used to refer to attributes and methods of the parent (super) class.

super.method();

RMIT UNIVERSITY

Method Overloading

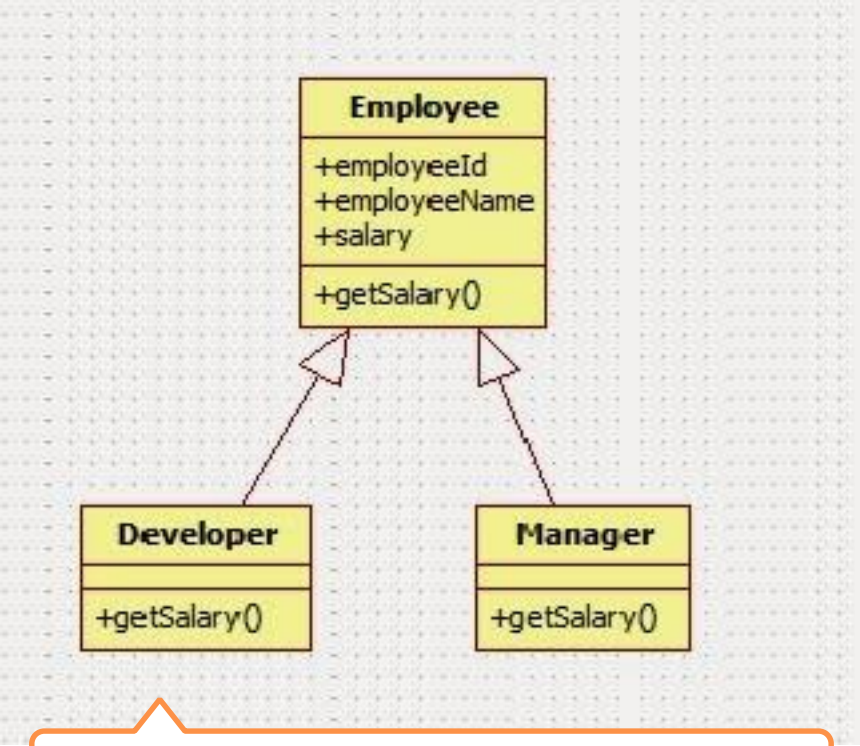
An instance method in a class is said to be overloaded if its signature has the same name, but it different numbers and/or type of parameters and return type

In the previous examples, the class constructors were in fact overloaded.
The same could be said of basic mathematical operators which can operate on integers, float or double.

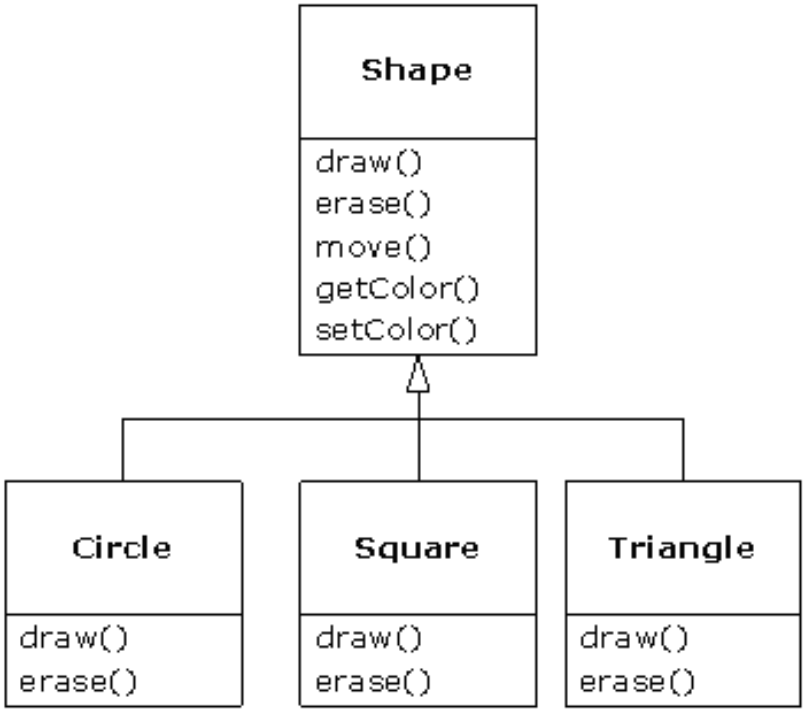
The core idea is that some methods are useful in slightly different contexts (different parameter and return types).

Method Overriding

Method overriding is similar to **overloading** except involves methods in different classes and in an inherited relationship**.**



The **salary( )** for the Manager & Developer will be different algorithms to each other & the general Employee of the parent class



The **draw()** method for each child class will be specialised than the more general method of the parent

http://www.java2blog.com/2014/06/method-overriding-in-java.html

http://www.linuxtopia.org/online_books/programming_books/thinking_in_java/TIJ303_005.htm

Method Overriding

An instance method (non static) in a subclass with the same signature (name, plus the number and the type of its parameters) and return type as an instance method in the superclass **overrides** the superclass's method.

The ability of a subclass to override a method allows a class to inherit from a superclass whose behaviour is "close enough" and then to modify behaviour as needed.

RMIT UNIVERSITY

Method Overriding

Example: The Person + Student class both had an writeOutput( ) method with no arguments

```
s.getName();          //call a method in the parent class
s.writeOutput() ;     //call own method from own (child) class
```

When an object of the **Student** class (child) calls the method which one is performed? The Person & Student class will have different code in these methods so it is important to know which one will be performed.

Java will call the child method. This is because it should be more specialised or relevant to the Student object. So, the child method always overloads the matching parent method.

Remember, a parent class is for the common data and the child classes contain the more specialised attributes and methods that uniquely model the child entities characteristics and behaviour.

RMIT UNIVERSITY

comparing the 2x Output( ) methods

Person (parent) class method

```
public void writeOutput( )
{
   System.out.println("Name: " + name);
}
```

The methods from the Person / Student example (shown here) are simplistic but show how you would expect more detail and specialisation in the child method.

Student (child) class method

```
public void writeOutput( )
{
   System.out.println("Name: " + getName( ));
   System.out.println("Student Number: " + studentNumber);
}
```

This method displays attributes from both classes

RMIT UNIVERSITY

Method Overriding

What if you wanted to call the parent overridden method and the child method? You may have some general processing done by the parent and then want to run the more specialised processing immediately afterwards.

No problem. Use super in the child method as shown below.

```
student203.writeOutput( );    //call method
```

```
public void writeOutput()      //method in child class
{
        super.writeOutput();     //call the overridden parent method
    System.out.println("School is " + school);

}
```

```
Name : Bazza
Age : 10
School is Cranbourne
```

output is super & child method combined

RMIT UNIVERSITY

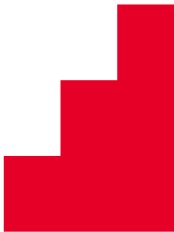
The `final` Modifier

If the modifier `final` is placed before the definition of a *variable*, then that variable is a constant. In other words it cannot be changed.

If the modifier `final` is placed before the definition of a *method*, then that method may not be redefined in a derived class. In other words it cannot be changed by providing a new version.

It the modifier `final` is placed before the definition of a *class*, then that class may not be used as a base class to derive other classes. In other words you cannot change the class by creating other subtypes.

RMIT UNIVERSITY



protected access

- protected provides additional behaviour to private and public

- If an instance variable is declared protected it can be accessed by methods of that class, its subclasses and all other classes within the same package (or directory).

- However, it cannot be accessed by other classes (those outside the package)

- The next code sample illustrates the differences across access specifiers.

RMIT UNIVERSITY

protected / private / public - access compared

```
class A {
…
      private int x;
      protected int y;
      public int z;

      void increment1() {
            x++; // valid
            y++; // valid
            z++; // valid
      }
}
class B extends A    //this class extends class A
{
      void increment2() {
            x++; // invalid
            y++; // valid
            z++; // valid
      }
}

class SomeOtherClass { // not in same package as class A

      void increment3() {
…
            A a = new A(…);
            a.x++; // invalid
            a.y++; // invalid
            a.z++; // valid
      }
}
```

Why are these declarations valid?

Why some valid and invalid?

Why some valid and invalid?

RMIT UNIVERSITY

Question/practice time – Part 2!

Implement the full person/student/local and international student class hierarchy tree

Implement all specified variables and getters/setters as needed

Implement an overridden **void writeOutput()** method in all classes specified, leveraging the base/sub class relationships and the **super** keyword

Create a Class object that contains a main method, in which 30 students are instantiated and added to an array of students.

Discussion: How would you implement an interface to input data for any kind of student? How would you write a method to read the 20 students records interactively on the command line?

RMIT UNIVERSITY

Summary:

- A derived class is obtained from a base class by adding instance variables and methods. The derived class inherits all public instance variables and public methods that are in the base class.
- When defining a constructor for a derived class, your definition should call a constructor of the parent class using super. If you do not make an explicit call, Java will automatically call the default constructor of the base class.
- Method overloading – redefining a method in a given class with same name but different parameters and/or return types
- Method overriding - redefine a method from a base/parent class so that it has a different definition in the derived class.
- Private instance variables and private methods of a base/parent class cannot be accessed directly by name within a derived class (but you can access protected variables and methods!)
- You can assign an object of a derived class to a variable of any ancestor type, but not the other way around.
- In Java, every class is a descendant of the predefined class Object. So every object of every class is of type Object, as well as the type of its class and any other ancestor classes.

Static

Static Variables

- Static means “pertaining to the class in general”, not to an individual object
- A variable may be declared (outside of a method) with the static keyword:

```
static int numTicketsSold;
```

- There is only **one** variable **numTicketsSold** for the class, not one per object!!!
- A static variable is shared by all instances
- All instances will be able read/write it
- A static variable that is public may be accessed using a *ClassName.AttributeName* notation e.g., **`Math.PI`**

Static Methods

- A method may be declared with the static keyword
- Static methods live at class level, not at object level
- Static methods access static attributes and methods, but not instance ones - how could it choose which one?

```
public static int getNumTicketsSold()
{
 return numTicketsSold;
}
```

- A static method that is public can be accessed using a ClassName.methodName(args) notation

```
double result = Math.sqrt(25.0);
int numSold = Ticket.getNumberSold();
```

RMIT UNIVERSITY

Example: Ticket

```
public class Ticket{
  private static int numTicketsSold = 0; // shared
  private int ticketNum; // one per object

  public Ticket(){
    numTicketsSold++;
    ticketNum = numTicketsSold;
  }
  public static int getNumberSold() {
    return numTicketsSold;
  }

  public String getInfo(){
    return "ticket # " + ticketNum + "; " +
            numTicketsSold + " ticket(s) sold.";
  }
}
```

RMIT UNIVERSITY

Exercise: Create TicketDriver to test the Ticket class.

```
public class TicketDriver{
   public static void main(String args[]){
       // print the number of tickets sold
       System.out.println("Tickets Sold :" + Ticket.getNumberSold());
       // Create ticket object t1
       Ticket t1 = …..
       // print the info of ticket t1
       System.out.println(……

       // print the number of tickets sold
       // Create a ticket object t2
       // print the info of ticket t2
       // print the number of tickets sold
       // print the info of ticket t1

    }
}
```

RMIT UNIVERSITY

Static context

To have a standalone Java Application we need a

```
public static void main(String args[])
```

method

The main method belongs to the class in which it is written

It must be static because, before your program starts, there *aren't any objects* to send messages to

This is a static context (a class method)

You can send messages to objects, *if* you have some objects: `d1.bark();`

You *cannot* send a message to yourself, or use any instance variables - this is a static context, not an object

Non-static variable cannot be referenced from a static context

RMIT UNIVERSITY

When to use static

A variable should be static if:

It logically describes the class as a whole

There should be only one copy of it

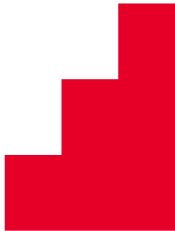
It doesn't violate the principle of cohesion

A method should be static if:

It does not use or affect the object that receives the message (it uses only its parameters)

In other words it is a utility method that does not manage state.

RMIT UNIVERSITY



Static Rules

static **variables and methods belong to the class in general, not to individual objects**

The absence **of the keyword *static* before non-local variables and methods means *dynamic* (one per object/instance)**

A dynamic method can access all dynamic *and* static attributes and methods in the same class

A static method can not access a dynamic attribute or method (because there is no obvious semantic to select which instance should be referenced)

RMIT UNIVERSITY