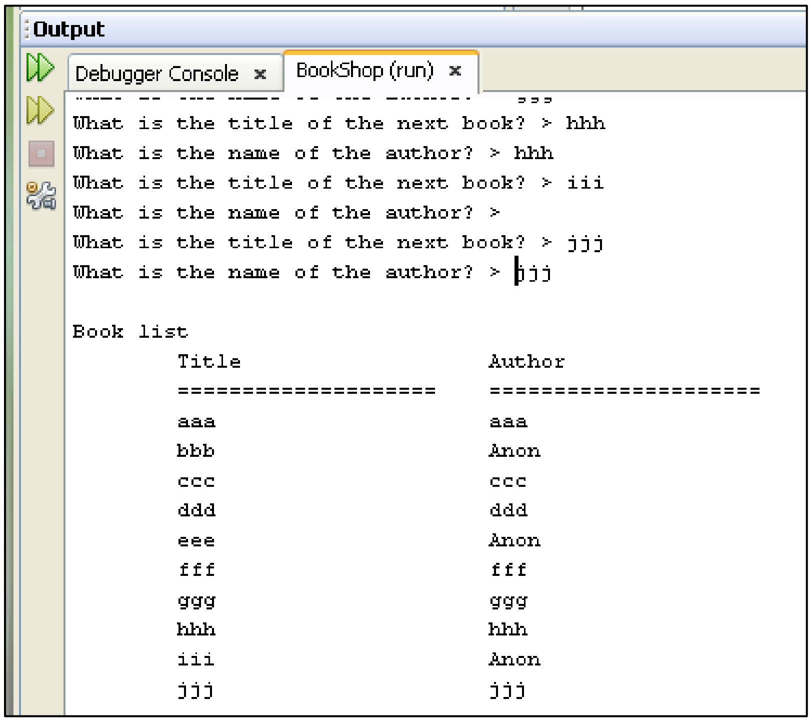
# Practical session 1

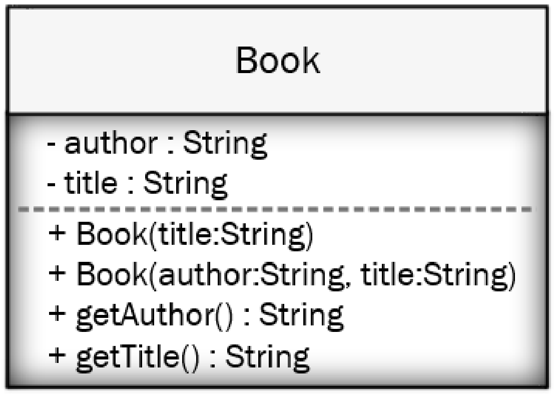
**This work should be completed before the next lecture.**

## Task 1: Book shop

Write an application that inputs and stores the titles and authors of ten books, and then outputs the book details to the console window. Use the UML class diagrams and accompanying pseudo-code given below. Follow them precisely. Your output should have a format similar to this screen shot.



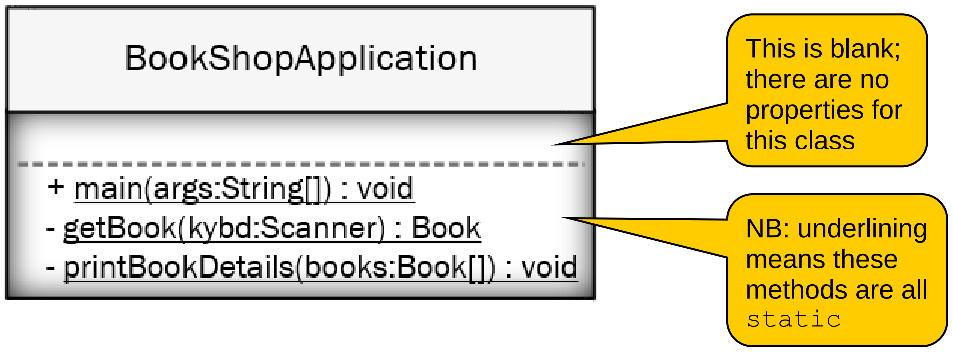
```
What is the title of the next book? > hhh
What is the name of the author? > hhh
What is the title of the next book? > iii
What is the name of the author? >
What is the title of the next book? > jjj
What is the name of the author? > jjj

Book list
        Title                   Author
        ====================    =====================
        aaa                     aaa
        bbb                     Anon
        ccc                     ccc
        ddd                     ddd
        eee                     Anon
        fff                     fff
        ggg                     ggg
        hhh                     hhh
        iii                     Anon
        jjj                     jjj
```



### Pseudo-code for Book

```
Book(title:String)
    this.title = title
    author = "Anon"
```

All other methods have expected behaviour


BookShopApplication
+ main(args:String[]) : void
- getBook(kybd:Scanner) : Book
- printBookDetails(books:Book[]) : void
This is blank; there are no properties for this class
NB: underlining means these methods are all static

**Pseudo-code for BookShopApplication**

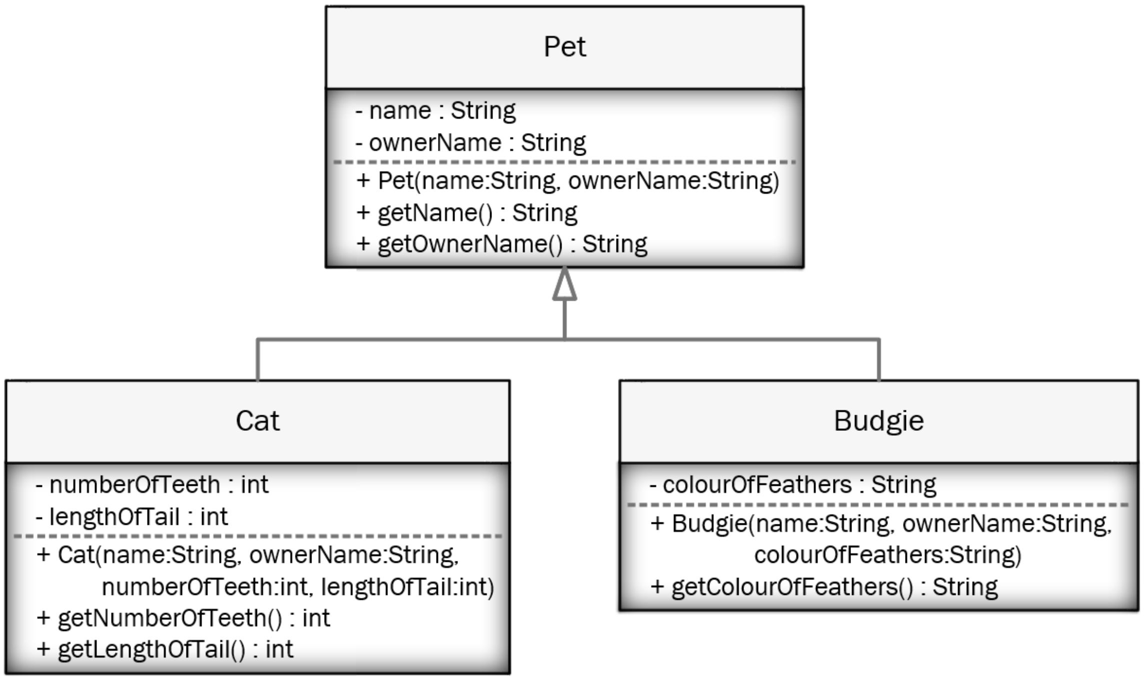
```
main(args:String[]) : void
     kybd = a scanner for reading from the keyboard;
     create an array called books to hold 10 Book objects

     for i = 0, books.length do
          books[i] = getBook(kybd)
     end for
     printBookDetails(books)


getBook(kybd:Scanner) : Book
     prompt = "What is the title of the next book? > "
     read title using kybd

     prompt = "What is the name of the author? > "
     read author using kybd

     if author is blank
          create new Book with title
     else
          create new Book with title and author
     end if

     return newly created Book


printBookDetails(books:Book[]) : void
     write "Book list"
     write "Title   Author"
     write "=====   ======"

     for i=0, books.length do
          write title and author of books[i]
     end for
```

**Portfolio requirements:**

- The NetBeans project for this completed task

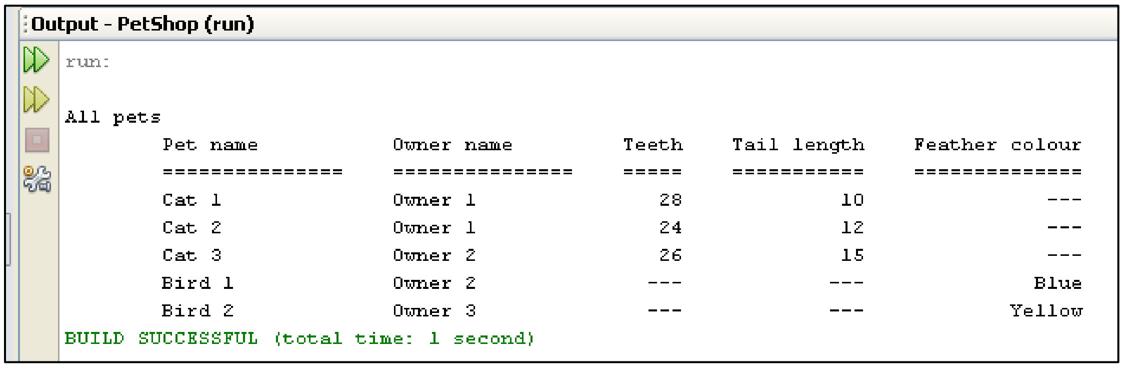
## Task 2: Pet shop with inheritance

Convert the UML class diagrams shown below into Java code. Be precise.


Pet
- name : String
- ownerName : String
+ Pet(name:String, ownerName:String)
+ getName() : String
+ getOwnerName() : String
Cat
- numberOfTeeth : int
- lengthOfTail : int
+ Cat(name:String, ownerName:String, numberOfTeeth:int, lengthOfTail:int)
+ getNumberOfTeeth() : int
+ getLengthOfTail() : int
Budgie
- colourOfFeathers : String
+ Budgie(name:String, ownerName:String, colourOfFeathers:String)
+ getColourOfFeathers() : String

Hint: Remember to ensure that the constructor methods in the subclasses call the superclass's constructor method.

Add to your project a Java class called `PetShopApplication`. The `main()` method should create three `Cat` objects and two `Budgie` objects. Next, the `main()` method should output the details of all pets to the console window in a tabulated format, similar to the illustration below.



Output - PetShop (run)

```
run:

All pets
        Pet name            Owner name          Teeth    Tail length    Feather colour
        ===============     ===============     =====    ===========    ==============
        Cat 1               Owner 1                28             10               ---
        Cat 2               Owner 1                24             12               ---
        Cat 3               Owner 2                26             15               ---
        Bird 1              Owner 2               ---            ---              Blue
        Bird 2              Owner 3               ---            ---            Yellow
BUILD SUCCESSFUL (total time: 1 second)
```

**Portfolio requirements:**

- The NetBeans project for this completed task

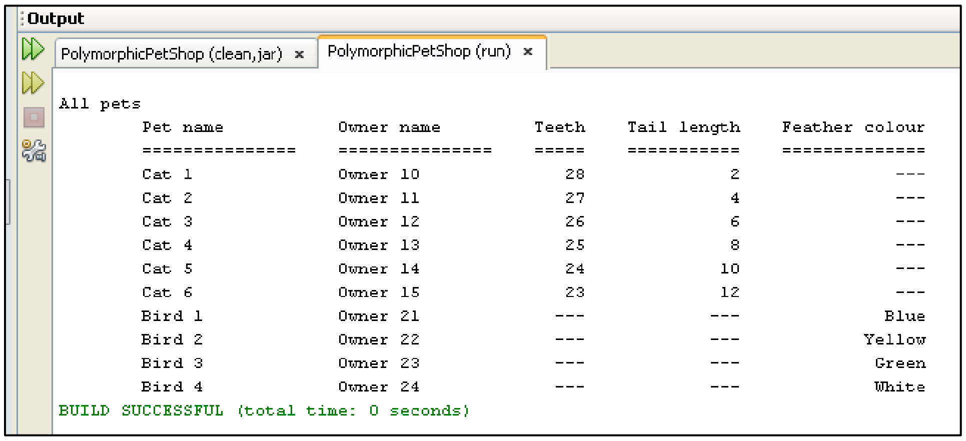
## Task 3: Pet shop with polymorphism

Make a copy of your NetBeans project from Task 2 and modify it so that polymorphism is used.

Instead of storing the pets in separate variables, use an array.

In the `PetShopApplication` class, add a method called `printPetDetails()`, which outputs to the console window the contents of the `pets` array in a tabulated format, similar to the illustration below. The `pets` array must be passed to the method as a parameter. Use the `instanceof` operator as shown in the `main()` method, `InheritanceApplication` class, `PersonInheritance3` project from the lecture.

The `main()` method should create six `Cat` objects and four `Budgie` objects, storing them all in the `pets` array, and then call the `printPetDetails()` method, passing the `pets` array as the parameter.



```
All pets
        Pet name              Owner name            Teeth    Tail length    Feather colour
        ===============       ===============       =====    ===========    ==============
        Cat 1                 Owner 10                 28              2               ---
        Cat 2                 Owner 11                 27              4               ---
        Cat 3                 Owner 12                 26              6               ---
        Cat 4                 Owner 13                 25              8               ---
        Cat 5                 Owner 14                 24             10               ---
        Cat 6                 Owner 15                 23             12               ---
        Bird 1                Owner 21                ---            ---              Blue
        Bird 2                Owner 22                ---            ---            Yellow
        Bird 3                Owner 23                ---            ---             Green
        Bird 4                Owner 24                ---            ---             White
BUILD SUCCESSFUL (total time: 0 seconds)
```

**Portfolio requirements:**

- The NetBeans project for this completed task

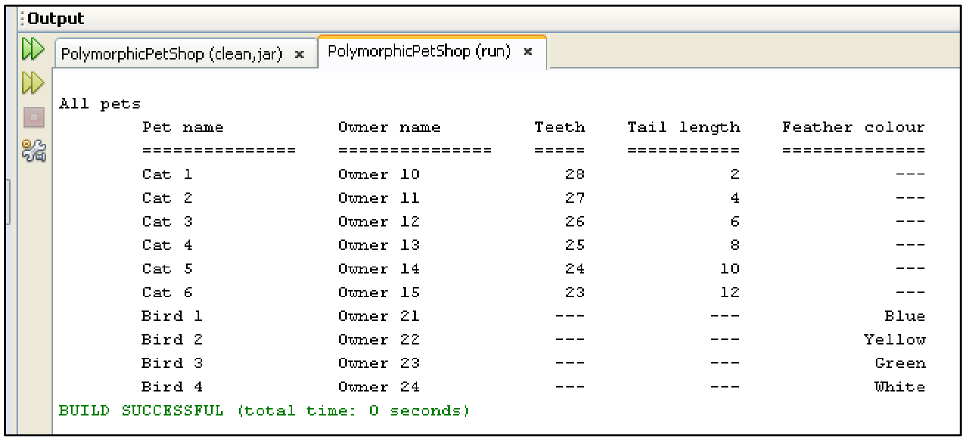
# Practical session 2

**This work should be completed before the next lecture.**

## Task 1: Pet shop with abstract `Pet`

Make a copy of your NetBeans project from Task 3 of Week 1.

Make the `Pet` class abstract. Correct any compilation errors that might be caused. The output should still look similar to this…



```
All pets
        Pet name            Owner name          Teeth    Tail length    Feather colour
        ===============     ===============     =====    ===========    ==============
        Cat 1               Owner 10               28              2               ---
        Cat 2               Owner 11               27              4               ---
        Cat 3               Owner 12               26              6               ---
        Cat 4               Owner 13               25              8               ---
        Cat 5               Owner 14               24             10               ---
        Cat 6               Owner 15               23             12               ---
        Bird 1              Owner 21              ---            ---              Blue
        Bird 2              Owner 22              ---            ---            Yellow
        Bird 3              Owner 23              ---            ---             Green
        Bird 4              Owner 24              ---            ---             White
BUILD SUCCESSFUL (total time: 0 seconds)
```

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 2: Pet shop with `equals()` and `hashCode()`

Make a copy of your NetBeans project from Task 1 and modify it as follows:

- In the `Pet` class, add two abstract methods to override the `equals()` and `hashCode()` methods.
- In the `Cat` class, override the `equals()` and `hashCode()` methods (you can use the “Insert code…” feature in NetBeans to generate them).
- In the `PetShopApplication` class, create another `Cat` object with the same data as one of the objects already created, and test to see if the two objects are equal. Output a message to confirm equality.
- In the `PetShopApplication` class, test two unequal `Cat` objects to see if they are equal. Output a message to confirm non-equality.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 3: Deck of Cards

In a new NetBeans project do the following:

- Create a `Card` class, that has two variables: Suit (Hearts, Clubs, Diamonds, Spades) and Rank (Ace, Two, Three, Four, … Jack, Queen, King). Use enums to represent Suit and Rank. Add suitable accessor (get) methods.
- Create a `Deck` class that has an array of 52 `Card` objects (one of each rank for each suit). Add an accessor method to return the array.
- Create a `CardApplication` class that creates a `Deck` object, gets, the array of `Card` objects, and outputs to the console window the entire deck of cards.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 4: Deck of Cards with local class

Make a copy of your NetBeans project from Task 3 and modify it as follows:

- In the `Deck` class, add a method that uses a local class to return an iterator for the array of `Card` objects (see lecture notes).
- Modify the `CardApplication` class so that it uses the iterator from the `Deck` object to output to the console window the entire deck of cards.

**Portfolio requirements:**

- The NetBeans project for this completed task

# Practical session 3

**This work should be completed before the next lecture.**

## Task 1: Team and Player output using JSON

Download from Blackboard the Team and Player classes.

Write an application that creates some `Team` and `Player` objects and then writes them as JSON strings to a file.

**Portfolio requirements:**

- The NetBeans project for this completed task
- The text file created by the program

## Task 2: Team and Player input using JSON

Write an application that reads JSON strings from the file produced in Task 1 and creates some `Team` and `Player` objects. Then, write the objects as JSON strings to a different file.

**Portfolio requirements:**

- The NetBeans project for this completed task
- The new text file created by the program

## Task 3: Deep cloning DVD

Write an application that has the following mutable classes:

- Person
  - First name – with accessor method
  - Last name – with accessor and mutator methods
  - getFullName() – returns first name and last name with a space between
  - Suitable constructor(s)
- DVD
  - Title – with accessor method
  - Lead actor (a Person object) – with accessor and mutator methods
  - Number of stars – with accessor and mutator methods
  - Suitable constructor(s)
  - toString() – returns the DVD title, lead actor's name, and number of stars as a suitably formatted String
- DVD application
  - main() – tests the functionality of DVD, including changing the number of stars in a DVD object

Modify the application so that the main() method invokes a deep clone of a DVD object. Output messages that show the deep clone has created different objects, as shown in the lecture notes.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 4: Immutable DVD

Copy the NetBeans project from Task 3.

Modify the DVD class so that its objects are immutable.

Modify the main() method so that it is still possible to change the number of stars for a DVD object.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Practical session 4

**This work should be completed before the next lecture.**

## Task 1: Team and Player using binary streams

Copy your NetBeans project from Task 2, 3 or 4 of Week 3.

Instead of reading and writing JSON strings to file using text streams, modify the application so that it reads and writes objects to file using binary streams.

**Portfolio requirements:**

- The NetBeans project for this completed task
- The binary data file created by the program

## Task 2: Catch up

Use the rest of your time this week to catch up on any tasks that you have not yet completed from all previous weeks.

# Practical session 5

**This work should be completed before the next lecture.**

## Task 1: Comparing `DVD` objects

Copy your NetBeans project from Task 3 of Week 3.

In the `DVD` class, add an integer field called `id`, with a get method.

Delete all statements from the `main()` method of the application, and any methods in the same class that are called from `main()`.

Like we did in the lecture demonstration, create multiple collections of `DVD` objects using:

- Array
- ArrayList
- Sorted ArrayList in natural order (number of stars, then title, then lead actor)
- Sorted ArrayList in lead actor order (leader actor, then title, then number of stars)
- HashMap (use the DVD's id as the key)
- TreeMap (use the DVD's id as the key)
- TreeSet in natural order (number of stars, then title, then lead actor)
- TreeSet in title order (title, then leader actor, then number of stars)

Print the contents of each collection.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 2: Using the `ArrayDeque` class

Copy your NetBeans project from Task 1.

Add to the application class an `ArrayDeque` collection of `DVD` objects. Using iterators, output its contents, first from front to back, and then from back to front.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 3: A generic `Stack` class

In a new NetBeans project, write a generic `Stack` class. The methods for a stack are:

- `void push(T item)`
  adds `item` to the top of the stack
- `T pop()`
  removes the top `item` from the stack and returns it
- `boolean isEmpty()`
  returns `true` if there are no items in the stack, and `false` otherwise

Make two versions of the `Stack` class:

1. Use an ArrayList to hold the items
2. Use an ArrayDeque to hold the items

In the `main()` method of the application:

- Create a `String` version, a `Person` version, and a `DVD` version of both `Stack` classes
- Push three items to each stack and make four attempts to pop an item from each stack
- Ensure the output to the console shows all push and pop attempts and reports all errors

**Portfolio requirements:**

- The NetBeans project for this completed task

# Practical session 6

**This work should be completed before the next lecture.**

## Task 1: A generic `Stack` class with custom exceptions

Copy your NetBeans project from Task 3 of Week 5.

Modify the generic `Stack` class as follows:

- Make the collection of objects an array with 5 places instead of an `ArrayList` or an `ArrayDeque`
- Throw a `FullStackException` custom exception when `push()` is called on a full stack
- Throw an `EmptyStackException` custom exception when `pop()` is called on an empty stack

In the `main()` method of the application:

- Create a `String` version and a `Person` version of the `Stack` class
- Push six items to each stack and ensure the `FullStackException` custom exception is thrown
- Pop six attempts item from each stack and ensure the `EmptyStackException` custom exception is thrown
- Ensure the output to the console shows all push and pop attempts and reports all errors

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 2: An auto-closeable generic `Stack` class

Copy your NetBeans project from Task 1.

Make the generic `Stack` class an auto-closeable resource. In the `close()` method, empty the stack.

In the `main()` method of the application:

- Use one `try`-with-resources statement to open and close both versions of the `Stack` class
- Push three items onto each stack inside the `try`-with-resources statement

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 3: Team and Player using `try`-with-resources statements

Copy your NetBeans project from Task 1 of Week 4.

Modify the code so use auto-closeable resources in `try`-with-resources statements.

**Portfolio requirements:**

- The NetBeans project for this completed task
- The binary data file created by the program

## Task 4: Catch up

Use the rest of your time this week to catch up on any tasks that you have not yet completed from all previous weeks.

# Practical session 7

**This work should be completed before the next lecture.**

## Task 1: Parameterised `DVD` supplier

Create a new NetBeans project.

Copy into the new project your `DVD` class from Task 1 of Week 5.

Following the concepts illustrated in the `IntegerArraySupplier` class, write a class that is a parameterised `Supplier` for your `DVD` class. You must include a constructor method that takes the title, lead actor and number of stars for the DVD to be created.

In the application class, create a `Supplier` that uses the DVD `Supplier` to create an `ArrayList` of 10 DVD objects.

In the `main()` method of the application class:

- create an `ArrayList` of 10 DVD objects
- use a `Consumer` to print the contents of the `ArrayList`

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 2: Parameterised `Predicate`

Copy your NetBeans project from Task 1.

Following the concepts illustrated in the `IntegerArraySupplier` class, add a class that is a parameterised `Predicate` that returns `true` if the DVD title starts with a letter in the range, `c1` (inclusive) to `c2` (exclusive), given to the constructor

Modify the `main()` method of the application class as follows:

- Use your `Predicate` class to print the details of the DVD objects whose title starts with a letter in the range A to D
- Use your `Predicate` class to print the details of the DVD objects whose title starts with a letter in the range P to W

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 3: Use a `Function`

Copy your NetBeans project from Task 2.

In the `main()` method of the application class:

- Write a `Function` that uses your `Predicate` class to return the number of DVD objects whose title starts with a letter in the range A to D; output that number
- Write a `Function` that uses your `Predicate` class to return the number of DVD objects whose title starts with a letter in the range P to W; output that number

**Portfolio requirements:**

- The NetBeans project for this completed task

# Practical session 8

**This work should be completed before the next lecture.**

## Task 1: Sports Day

Write a program to help a teacher decide who will run in four different events at a primary school sports day. The four events and their participants are:

| | |
|---|---|
| • Sack race: | pupils whose names begin with 'a' |
| • Egg and spoon: | pupils aged 5, 6 or 7 |
| • 3-legged race: | pupils in the first half of the register |
| • Obstacle race: | pupils in the second half of the register |

The user should input the number of pupils followed by their names and ages. Store the names and ages in an `ArrayList` of `Person` objects. Use Java streams to print who will be in each race.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 2: Salaries, high to low

Write a program to print employees in order (high to low) of salary. The user should input the number of employees followed by their names, department names and salaries. Store the details in an `ArrayList` of `Employee` objects. Use a stream pipeline to print the list of employees in ascending order of name within descending order of salary.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 3: Salaries, high to low, with a file

Copy your NetBeans project from Task 2.

Instead of having the user input employee details, store them in a file. Read the details from the file as a stream.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 4: Unique salaries

Copy your NetBeans project from Task 3.

Modify the program so that it also prints a list of the distinct salary values.

**Portfolio requirements:**

- The NetBeans project for this completed task

# Practical session 9

**This work should be completed before the next lecture.**

## Task 1: Sports Day collections

Copy your NetBeans project from Task 1 for Week 8.

Modify the program so that it uses Java streams to collect the participants of each race into `List`s. Print the contents of each list.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 2: Average salary

Copy your NetBeans project from Task 3 for Week 8.

Modify the program so that it uses a stream reduction to calculate the average salary of all employees. Print the average salary after printing the list of employees in ascending order of name within descending order of salary.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 3: Employee collection

Copy (again) your NetBeans project from Task 3 for Week 8.

Modify the program so that all employees in a department are printed. The user inputs the name of the department. Use a collector for the printing, as shown in the lecture demonstration.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 4: Factorial with streams

Use streams to calculate and print the factorial of a positive integer input by the user.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 5: Explore `Collectors`

In your independent study time, explore the `Collectors` class to become familiar with its other various methods.

Try writing some code that uses these other methods.

**Portfolio requirements:**

- The Java code you write for this task

# Practical session 10

**This work should be completed before the next lecture.**

## Task 1: Query the sample customer table

Download the lecture example from Blackboard.

Using NetBeans, modify the example so it queries the `Customer` table in the sample Java DB database and prints all data to the console.

Hint: Use NetBeans to explore the table.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 2: Insert a customer

Copy your NetBeans project from Task 1.

Add code to the application so that it prompts the user for information about a new customer and inserts the data into a new row in the `Customer` table. Query and print the data in the `Customer` table.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 3: Delete a customer

Copy your NetBeans project from Task 2.

Add code to the application so that it prompts the user for a customer id and deletes from the `Customer` table the row with that id. Query and print the data in the `Customer` table.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 4: VowelCount with database

### Step 1: Download a NetBeans project

Download the `VowelCount` NetBeans project from Blackboard, unzip the file, and open it in NetBeans.

Run the program and examine the code in the `main()` method and the `VowelCounter` class. Make sure you understand it. Ask for help, if needed.

### Step 2: Create a database table

In your database, create a table called `VowelTally`, which is described as follows:

| Column name | Type | Length | Comments |
|---|---|---|---|
| Vowel | String | 1 | Not null, an uppercase vowel |
| Tally | Integer | | Not null, initially zero |

Insert five rows (one for each vowel) with Tally initialised to zero.

### Step 3: Modify the application to use your database

Write the required code in each method of the `VowelTallyGateway` class.

Modify the main() method in the application so that it uses the methods in the `VowelTallyGateway` class to:

- Add the number of vowels in the user's string to the tally in the database
- Get all tallies from the database and print them

**Portfolio requirements:**

- The NetBeans project for this completed task

# Practical session 11

**This work should be completed before the next lecture.**

## Task 1: Design patterns in your customer application

Copy your NetBeans project from Task 3 of Week 10.

Refactor your code so that the following design patterns are used:

- Singleton
- Factory
- Builder
- Null Object
- Table Data Gateway
- Command

You can introduce the patterns in any order you choose.

**Portfolio requirements:**

- The NetBeans project for this completed task

# Practical session 12

**This work should be completed before the next lecture.**

## Task 1: Unit testing in your customer application

Copy your NetBeans project from Task 1 of Week 11.

Write JUnit tests to ensure the following:

- Singleton
  - Two calls to `getInstance()` return the same object
- Factory
  - An object of the correct class is returned by the factory method for each value (including invalid values) of the code
- Builder
  - The builder returns a built object with the correct values

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 2: Unit testing the Lecture11Demo

Download from Blackboard the `Lecture11Demo-Complete` NetBeans project.

Write JUnit tests to ensure the following:

- Copy
  - After creation, state has the correct value
  - After borrowing, state has the correct value
  - After returning, state has the correct value
  - Borrowing a copy twice without having a return operation between throws the correct exception
  - Returning a copy twice without having a borrow operation between throws the correct exception
- Book
  - The constructor throws the correct exception with the correct message when title, author and ISBN are all blank (i.e. the empty string, “”)
  - When there are multiple copies of a book, `findCopy()` returns the correct object
  - When a copy does not exist, trying to find it throws the correct exception
  - Can add five copies
  - Cannot add more than five copies
  - `addCopy()` returns `true` and `false` appropriately

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 3: Write a test plan

Given the following problem description, write the requirements, test cases and test plan.

**Problem**

A simple calculator class is needed that permits the user to add, subtract, multiply, and divide decimal point values. It operates on two operands at a time. It also maintains a single memory to which a value can be added (i.e. increasing the current value in memory). The value in memory can be retrieved and also reset to zero. The calculator determines the result of applying the currently selected operator to the current values of the two operands. The result can be retrieved. The calculator can be cleared which resets all values to zero, and the operator to space.

**Portfolio requirements:**

- A Word document containing the requirements, test cases and test plan

## Task 4: Implement a test plan

Create a NetBeans project, and implement your test plan from Task 3 as a set of JUnit tests.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 5: Implement an application that satisfies JUnit tests

In your NetBeans project from Task 4, write a Java application that passes all your JUnit tests.

**Portfolio requirements:**

- The NetBeans project for this completed task

# Practical session 13

**This work should be completed before the next lecture.**

## Task 1: Prepare to use threads

Create a NetBeans project for this task.

Write a class `Message` that contains the following method, which outputs a string of text (`msg`) a number of times (`num`):

```
public void run() {

      for (int i = 0; i < num; ++i) {

            System.out.println(msg + " " + i);

      }

}
```

The string to output (`msg`) and the number of times (`num`) are instance variables (attributes) that must be initialised in the `Message` constructor using parameters values.

Write a class with a `main()` method that instantiates the `Message` class several times with different messages and different numbers of times, and calls the `run()` method for each object. Do NOT use threads at this stage.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 2: Using threads

Copy your NetBeans project from Task 1.

Modify your `Message` class to support threads. The `run()` method should execute when the thread starts. Modify your `main()` method so that each instance of the `Message` class is run on a separate thread. Is there any difference in output from the testing in Task 1? Why?

Now call `Thread.sleep(100)` to the loop within the `run()` method to make the loop take longer than a single time slice. Repeat the testing. What is the difference in output? (You may need to increase the sleep time gradually until you see a change in the output.) Can you explain what is happening?

Now modify the `Thread.sleep()` method call so that the time to sleep is chosen at random. Repeat the testing. What is the difference in output? Can you explain what is happening?

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 3: Producer-Consumer

Download from Blackboard the `Lecture13Demo` NetBeans project.

Add two more classes, `Producer` and `Consumer`, which extend `Thread`. The constructor for each class should take a reference to a `Buffer` object.

The `Producer`'s `run()` method should have a loop to add the numbers 1 to 15 to the `Buffer` object. After each number is added, its thread should sleep for a short interval.

The `Consumer`'s `run()` method should have a loop to retrieve the next number from the `Buffer` object, then sleep for a short interval.

Add another class, `ProducerConsumerMain`, containing the following main method:

```
    public static void main(String[] args) {
        Buffer b = new Buffer(2);
        Producer p1 = new Producer(b, 1);
        Producer p2 = new Producer(b, 2);
        Consumer c1 = new Consumer(b, 1);
        Consumer c2 = new Consumer(b, 2);
        p1.start();
        p2.start();
        c1.start();
        c2.start();
    }
```

Without changing any of the code in the `Buffer` or `ProducerConsumerMain` classes, implement your `Producer` and `Consumer` classes so that your program has output similar to the following (each run will be different because of the random sleep intervals):

```
Producer 1 added 1 to buffer:[1]
Consumer 2 retrieved 1 from buffer: []
Consumer 1 attempting to remove from empty buffer - wait
Producer 2 added 1 to buffer:[1]
Consumer 1 retrieved 1 from buffer: []
Producer 2 added 2 to buffer:[2]
Producer 1 added 2 to buffer:[2, 2]
Producer 1 attempting to add to full buffer - wait
Producer 2 attempting to add to full buffer - wait
Consumer 1 retrieved 2 from buffer: [2]
...
```

**Portfolio requirements:**

- The NetBeans project for this completed task

## Practical session 14

**This work should be completed before the next lecture.**

## Task 1: Use `Callables` and an `ExecutorService`

Copy your NetBeans project from Task 3 of Week 13.

Change the `Producer` and `Consumer` classes into `Callables` instead of sub-classes of `Thread`.

Modify the `main()` method so that it uses an `ExecutorService` to run instances of the `Producer` and `Consumer` classes.

**Portfolio requirements:**

- The NetBeans project for this completed task

## Task 2: Catch up

Use the rest of your time on the module this week to complete any previous tasks you have not yet finished.