proprep

Workbook

Table of Contents

Object Oriented Programming

Object Orientation

Questions

1) Create a new project in BlueJ, and a class within called `Name`

a. Create two private instance variables of `String` datatype called `first` and `last`

b. Create a public method called `setFirst` which takes a `String` argument and stores it into the variable `first` and returns no value

c. Create a public method called `setLast` which takes a `String` argument and stores it into the variable `last` and returns no value

d. Create a public method called `getFullName` which takes no arguments but returns a `String`. When called, `getFullName` should return the contents of `first` and `last` concatenated with a space between them

e. Test your code by creating an object from the class on the BlueJ object workbench:

i. Call `setFirst` with the value `"Margaret"`

ii. Call `setLast` with the value `"Hamilton"`

iii. Call `getFullName` which should return `"Margaret Hamilton"`

f. Create a second object from the same class and repeat the tests from part e) but set your own first and last name as the values. Double click each of the objects on the object workbench, one at a time and check the values of the instance variables

2) Create a class called `BankAccount` which will be used to store a bank account balance. Deposits to and withdrawals from the account can only be made with the provided interface methods.

a. Within the class, create a private instance variable balance of datatype `double` with initial value 0

b. Create three public methods within the `BankAccount` class:

   i. `deposit`, which accepts a single `double` argument and returns nothing

   ii. `withdraw`, which accepts a single `double` argument and returns a `boolean`

   iii. `getBalance`, which accepts no arguments but returns a `double`

c. For the `deposit` implementation, take the value passed into the method, and add it to the current value of the instance variable `balance`, storing the result back into `balance`

d. For the `getBalance` implementation, return the current value of `balance`

e. For the `withdraw` implementation,

   i. first check if subtracting the withdrawal amount from `balance` would give a negative result

   ii. if it does not, then subtract the withdrawal amount from `balance`, store the result back into `balance` and return `true`

   iii. if it would, then simply return `false` without reducing the balance

f. Test your code, by creating an object from the `BankAccount` class on the object workbench, and then:

   i. Add 1955.50 to the account by using the `deposit` method

   ii. Inspect the instance variable `balance` by double clicking the object on the object workbench

   iii. Add another 10.0 to the account using `deposit` again, and run `getBalance` which should return 1965.50

   iv. Withdraw 1000.50 using the `withdraw` method, which should return `true`

   v. Run `getBalance` again which should return 965.0

   vi. Attempt to withdraw 966 using `withdraw` which should return `false` and leave `balance` unchanged

3) Create a new class called `Clock`

a. Create three private integer instance variables within `Clock` called `hours`, `minutes`, and `seconds`, with initial values of `12`, `0` and `0` respectively.

b. Create a public method called `set` which accepts three parameters, `h`, `m` and `s` and returns nothing. The method should store the values in the three parameters into the `hours`, `minutes` and `seconds` instance variables.

c. Create a private method called `displayTime` which returns nothing and has no parameters. This should output the contents of the instance variables to display the time, e.g. `12:59:57`

d. Create another public method called `run` which advances `seconds` by 1 each time it is called.

   i. When `seconds` reaches 60, advance `minutes` by 1, reset `seconds` to 0

   ii. When `minutes` reaches 60, advance `hours` by 1, reset `minutes` to 0

   iii. When `hours` reaches 13, reset `hours` to 1

   iv. At the end of the `run` method, call the `displayTime` method

e. Test by creating an object from the `Clock` class on the BlueJ object workbench. Set the output of the terminal window to "clear screen on method calls" if it isn't already.

   i. Using `set`, set the time to 12 hours 59 minutes and 57 seconds

   ii. Inspect the instance variables to see that they have been correctly set by double clicking the object in the object workbench

   iii. Call the run method three times to see if the time wraps around to 01:00:00

f. Why does `displayTime` not appear in the list of methods when you right click on the object in the object workbench?

4) Create another class within the same project called `Clock24`

a. Produce a modified version of the clock from the previous exercise with a modified `run` method that keeps 24 hour time instead of 12 hour time

b. Add a private String attribute called `location`

c. Add a public method called `setLocation` which accepts a String parameter and returns no value. It should take the value passed in and store it in `location`

d. Modify the `displayTime` method so it shows the value of `location` before the time and prints a newline character at the end of the time

e. Test by creating three different objects from the `Clock24` class. Turn off the "Clear screen at method call" option in the BlueJ terminal window

   i. Set one clock to a time of 09:30:00 and location "`New York`"

   ii. Another clock should have a time of 13:30:00 and location "`London`"

   iii. The final one should have a time of 21:30:00 and location "`Tokyo`"

   iv. Use the `run` method on each of the three clocks to see if the time is correctly kept for each of the three individual clocks

5) Create a new class called `TeamMember` which will be used to store the name of a person, and a single task assigned to them

a. It should have two `String` instance variables, `name` and `task`
b. Create a method `setName` that allows the `name` variable to be set to a string passed in as a parameter and returns no value
c. Create another method `addTask` that allows a task to be passed in as a string and stores it into the `task` instance variable, but only if there has not been a task previously set (i.e. `task` == `null`). If a task is successfully added, the method should return `true`, otherwise it should return `false`
d. Create a method called `getTask` which
    i. Copies the task previously stored into a temporary variable
    ii. Stores `null` into the `task` instance variable
    iii. Returns the contents of the `name` variable, concatenated with a colon and space, concatenated with the previously stored task from the temporary variable, e.g. `"Bob: pack orders"`
e. Test your code by creating an object from the class, then
    i. Use `setName` to set the name to `"Bob"`
    ii. Use `addTask` to set a task of `"deliveries"` which should return `true`
    iii. Use `addTask` again to set a task of `"pack orders"` which should return `false`
    iv. Use `getTask` which should return the string `"Bob: deliveries"`
    v. Create another object from the class, with a name `"Sara"` and task of `"sales"`
    vi. Run getTask on this second object twice which should return "Sara: sales" the first time and then "Sara: null" the second time

6) Modify the code you wrote for the previous exercise so that multiple tasks can be assigned to a team member
   a. Change the instance variable `task` into an array of 5 strings
   b. Create a new instance variable `index` which is used as an index into the array and also keeps track of how many tasks have been assigned
   c. Modify `addTask` so it allows multiple tasks to be set and stored in the array, but only up to a limit of 5 tasks in total. If it is able to store a task the method should return `true`, otherwise it should return `false`
   d. Modify `getTask` so it only removes and returns the most recently added task in the array
      i. Update `index` if it is not zero already
      ii. Mark the item to be removed as `null`
      iii. Return the current item concatenated with name and a colon e.g. `"Bob: pack orders"`
      iv. If there are no tasks in the array simply return `name` and `": null"` concatenated
   e. Test as follows by creating an object from the class:
      i. Use `setName` to set the name to `"Bob"`
      ii. Use `addTask` to set a task of `"deliveries"` which should return `true`
      iii. Use `addTask` again to set a task of `"pack orders"` which should return `true`
      iv. Use `addTask` again to set a task of `"sales"` which should return `true`
      v. Use `addTask` again to set a task of `"warehouse"` which should return `true`
      vi. Use `addTask` again to set a task of `"customer service"` which should return `true`
      vii. Use `addTask` again to set a task of `"cleaning"` which should return `false`
      viii. Use `getTask` which should return the string `"Bob: customer service"`
      ix. Use `getTask` again which should return the string `"Bob: warehouse"`
      x. Continue until `"Bob: null"` is returned

Answer Key

To view the answers to these exercises please refer to the appropriate videos.

Classes

Questions

1) Create a new project and within it a class called `Student`

a. The class should have two `String` attributes `first` and `last`, and an `int` attribute `year`

b. Create a constructor that accepts two `String` parameters named `firstname`, `lastname` and one integer parameter named `enrolmentYear` and sets the instance variables to those values

c. Create a selector method called `getDetails` that returns the first name, last name and enrolment year all concatenated together with spaces in between

d. Test by creating objects and running `getDetails` using as input values:

i. `"Ben", "Chen", 2022` – returns `"Ben Chen 2022"`

ii. `"Anita", "Gill", 2023` – returns `"Anita Gill 2023`

2) Create a new class called `Car`

i. The class should have three `String` attributes: `manufacturer`, `model`, `registration`

ii. Create a constructor that accepts parameters for the car's manufacturer, model and registration, and sets the instance variables to these values

iii. Create an alternative constructor that accepts only two parameters, for manufacturer and model, and sets the instance variables to these values, and sets registration to "unknown"

iv. Create another constructor that accepts no parameters, and sets all three attributes in the object to "unknown"

v. Test by creating three objects:

i. For the first, use the arguments "Ford", "Mustang", "MUS T4NG"

ii. For the second use the arguments, "Neo", "Futuro"

iii. For the third, give no arguments.

iv. Double click on each object to confirm the contents of the attributes are what you would expect.

3) Create a new class called `Customer`

a. Create three attributes, a `String` called `fullName`, a `boolean` called `hasLogin` and an `int` called `noOfPurchases`

b. Create three different constructors

i. The first takes one argument, and the parameter should be called `fullName`. It should use this to set the `fullName` attribute to whatever has been passed in, and set `hasLogin` to `false`, and `noOfPurchases` to 0.

ii. The second takes two arguments, and the parameters should be called `fullName` and `hasLogin`. It should set the `fullName` and `hasLogin` attributes to the values passed in and `noOfPurchases` to 0.

iii. The third takes three arguments, and the parameters should be called `fullName` and `hasLogin` and `noOfPurchases`. It should set all the instance variables to the values passed in.

c. Create three objects from the class as follows:

i. With one argument "Bill Booker"

ii. With two arguments "Bob Baker" and true

iii. With three arguments "Beryl Brown", true, and 3

d. Double click each object to see if the constructors have correctly set the attributes to the values you expect.

4) Create a new class called `WageSlip` based on the class diagram below

WageSlip
-name: String -hours : int -hourlyRate : double -taxRate : double
+WageSlip(String, int, double) +WageSlip(String, int, double, double) +getPayGross() : double +getPayNet() : double

a. One of the constructors should accept a name, hours worked, and an hourly rate. It should set `taxRate` to 0.2 and the other attributes to the values passed in. The other constructor takes an extra parameter used to set the value of `taxRate` so all 4 values passed in are stored into the relevant attribute.
b. Implement `getGrossPay` so it returns the hourly rate multiplied by the number of hours worked
c. Implement `getNetPay` so it returns the gross pay with tax deducted according to the set tax rate, e.g. for a gross pay figure of 1000 a default `taxRate` of 0.2 will deduct 20% from the gross pay,
resulting in net pay of 800
d. Test by creating four different objects and run `getNetPay` and `getGrossPay` on each as follows:
    i. "Elly Ash", 25, 15 (gives gross: 375, net: 300)
    ii. "Karen Lopez", 35, 15 (gives gross: 525, net: 420)
    iii. "Ben Green", 35, 25, 0.4 (gives gross: 875, net: 525)
    iv. "Ken Li", 35, 25, 0.45 (gives gross: 875, net: 481.25)

5) Create a new class called `Kangaroo` based on the class diagram below

Kangaroo
-size : int -colour : string -age : int -asleep: boolean -count: int
+Kangaroo(int, string, int) +jump(int, char) +sleep() +wake() +isAsleep() : boolean +displayStatus() +getCount() : int

a. Use a constructor to set the values of the attributes `size`, `colour` and age to values passed in at instantiation, and `asleep` to `false`.
b. Note how count is underlined and is used to keep track of how many kangaroo objects have been instantiated. Add code to the constructor that will keep count of the number of kangaroos created.
c. Implement the jump method to display the message "Jumping: " concatentated with the values of the two parameters passed in, e.g. `"Jumping: 2 high, direction F"`
d. Implement `sleep` and `awake` methods to set the `asleep` attribute to the appropriate value.
e. Implement the `isAsleep` method so it returns a value indicating whether the kangaroo is currently asleep or not.
f. Implement `displayStatus` so that a message is displayed indicating its age, `colour` and `size` and whether the kangaroo is asleep or awake. For example `"This is a red kangaroo, size 1, age 3". The kangaroo is currently sleeping."`
g. Test by creating 3 kangaroo objects on the object workbench with the following values at instantiation:
   i. 1, "red", 3
   ii. 2, "grey", 5
   iii. 3, "brown", 4
   iv. Run `displayStatus` on all three objects

v. Run the `sleep` method on one of the kangaroos, and confirm the effect by running `isAsleep` and `displayStatus`
vi. Run `getCount` and confirm that 3 is returned

h. Modify the `getCount` method so it is a class method. When creating objects in the object workbench what has now happened to the `getCount` method?
i. How can we run `getCount` from within the BlueJ environment?

6) Create a class called `SimLauncher` which will make use of the `Kangaroo` class which you created in the previous exercise.

a. Create a public static method called `main` which returns no value but accepts an array of Strings, with a parameter name `args`
b. Inside the `main` method, create three instances of the `Kangaroo` class, called `kanga1`, `kanga2` and `kanga3`
c. Again, inside the `main` method run the `sleep` method on `kanga1`
d. Once more, inside `main`, run the `displayStatus` method on each of the three `Kangaroo` objects
e. Use the class method `getCount` to display a message saying how many kangaroos there are in the simulation.
f. Make sure you have activated the terminal window in BlueJ, then right click on the `SimLauncher` class, and select the `main` method. What happens, and why did no object get created on the object workbench?

Answer Key

To view the answers to these exercises please refer to the appropriate videos.

Inheritance

Questions

1) Create a new project and within it a class called `Person`
   a. Create three private attributes, `name`, `age` and `birthplace`
   b. Create a constructor which sets the values of each of these attributes to the values passed in as arguments
   c. Add a static `main` method to this class which creates a `Person` object called `person1` using the name "Ray", age 78 and place of birth "London"
   d. The second line of the `main` method should be:
   `System.out.println(person1);`
   e. Run the `main` method and note what appears in the terminal output
   f. Override the `toString` method that this class automatically inherits from the `Object` class. The method should return the attributes within the class as a concatenated string, e.g. "Name: Ray, age: 78, place of birth: London"
   g. Run the `main` method again. What is printed now and why?

2) Make use of the `Person` class from the previous exercise to:

a. Create two instances, `person1` and `person2` of a `Person` on the object workbench using the same attribute values for each of "Ray", 78 and "London"

b. Right click on `person1` and:

i. Select the "inherited from Object" option, which then presents a submenu from which you should select the `equals` method

ii. Type `person2` into the dialog box which will compare the `person1` object to the `person2` object for equality

iii. What is returned, and why?

c. Provide a new implementation in `Person` of the `equals` method that overrides the default version inherited from the `Object` class

i. The method should return `true` if the values stored in the instance variables are all the same in both objects, otherwise it returns `false`

ii. Note you will need to declare the `equals` method to accept an argument of type `Object`, and then use a type cast to create a temporary object of type `Person` to enable the code to compile correctly

d. Test by creating three instances of `Person`:

i. `person1`, with values "Ray", 78 and "London"

ii. `person2`, with values "Ray", 78 and "London"

iii. `person3`, with values "Dave", 78 and "London"

iv. Compare `person1` to `person2` using `equals` – should now return `true`

v. Compare `person2` to `person1` using `equals` – should also return `true`

vi. Compare `person3` to `person1` using `equals` – should return `false`

3) Create a class called `Item` as follows:

Item
#description : String #quantity : int
+Item(string, int) +getDescription(): String +getQuantity() : int +displayDetails()

a. Implement all the methods. `displayDetails` should print to the console a message as follows: `"Description: blue pen. In stock: 40"` depending on the contents of the `description` and `quantity` attributes.
b. Implement a child class, `SpecialItem` which inherits from `Item` but has an additional attribute called `deposit`, a `double`, which is used to store the value of a deposit that should be paid for specially ordered non stock items.
c. The constructor for `SpecialItem` should take an argument for `description`, plus another argument which contains the deposit value, and should always set `quantity` to `-1`
d. Implement a new version of `displayDetails` within `SpecialItem`, which instead of displaying the quantity in stock displays the message: `"Description: engraved pen. This special item requires a deposit before ordering of: 5.15"`
e. Test your code by creating two objects as follows:
   i. An object of type `Item`, with the description `"blue pen"` and quantity 40
   ii. An object of type `SpecialItem` with the description `"engraved pen"` and deposit value 5.15
   iii. Run `displayDetails` on each object to see if you get the expected output

4) Create a class called `Pen` which inherits from the `Item` class

a. The pen has an additional two attributes:
   i. A `String` called `colour`
   ii. A `boolean` called `isRetractable`

b. The constructor for `Pen` should take three arguments to set the attributes for colour, quantity and whether the pen is retractable or not. It should automatically set the `description` attribute to an appropriate value depending on the `colour` attribute, for example with a value of blue `description` will be set to `"blue pen"`, with a value of black `description` will be set to `"black pen"`

c. Override the `displayDetails` method within the Pen class so that it displays whether the pen is retractable or not, for example:

```
Description: black retractable pen. In stock: 999
```

d. Create another class called `PromoPen` which inherits from the `Pen` class

e. This class should have an additional attribute `message` which contains a promotional message

f. The constructor should accept an additional argument to set the `message` attribute

g. The `displayDetails` method should call the `displayDetails` method from the parent class, and then add an additional line of output showing the promotional message, for example:

```
Description: black retractable pen. In stock: 999
Message: webShoppy.com for your stationery needs 24/7
```

h. Test your code by creating objects as follows:
   i. A `Pen object`, with colour blue, retractable set to false and quantity 40
   ii. A `Pen object`, with colour red, retractable set to true and quantity 10
   iii. A `PromoPen` object with colour black, retractable set to true, quantity set to 999 and message set to "webShoppy.com for your stationery needs 24/7"
   iv. Run `displayDetails` on each object to see if you get the expected output

5) Create a class called `Clothing` as follows which inherits from the `Item` class created in a previous exercise (note that attributes and methods inherited from `Item` are not shown in this class diagram):

Clothing
#size: String #colour: String
+Clothing(string, string, string, int) +getSize(): string +getColour(): string +displayDetails()

a. Implement all the methods in the Clothing class . The implementation of `displayDetails` should show the size and colour of the item as follows: `"Description: t-shirt. Size: medium. Colour: white. In stock: 7"`

b. Create another class, `CustomTshirt` which inherits from the `Clothing` class, and adds an extra attribute called `text`.

   i. Ensure that the constructor for `CustomTshirt` accepts an argument for the `text` attribute and initialises it to this value. There is no need for a description argument, ensure the constructor always sets the description attribute to `"Custom t-shirt"`

   ii. Implement an extra selector method for the new attribute

   iii. `displayDetails` for `CustomTshirt` should call the `displayDetails` method from the parent class, and should then output on a separate line the customised text, e.g.: `"Customisation: Climate Action Now!"`

c. Create another class `CustomHoody` which inherits from the `Clothing` class, and adds an extra two attributes: `frontText` and `backText`.

   i. There is no need for a description argument, ensure the constructor always sets the description attribute to `"Custom hoody"`

   ii. The constructor will not need any extra arguments to set the values of `frontText` and `backText`. Instead provide a modifier method `setText` which takes two arguments and sets `frontText` and `backText` to the values passed in.

   iii. `setText` should ensure that the text is no longer than 25 characters for either of the two values

iv. If `setText` was able to set the text for both attributes, it returns the boolean value `true`, otherwise it returns `false`

v. `displayDetails` for `CustomHoody` should call the `displayDetails` method from the parent class, and should output on a separate line the customised text, e.g.: `"Customisation front: Cranhurst College CS. Customisation back: Class of 2022"`

d. Test by creating objects as follows:

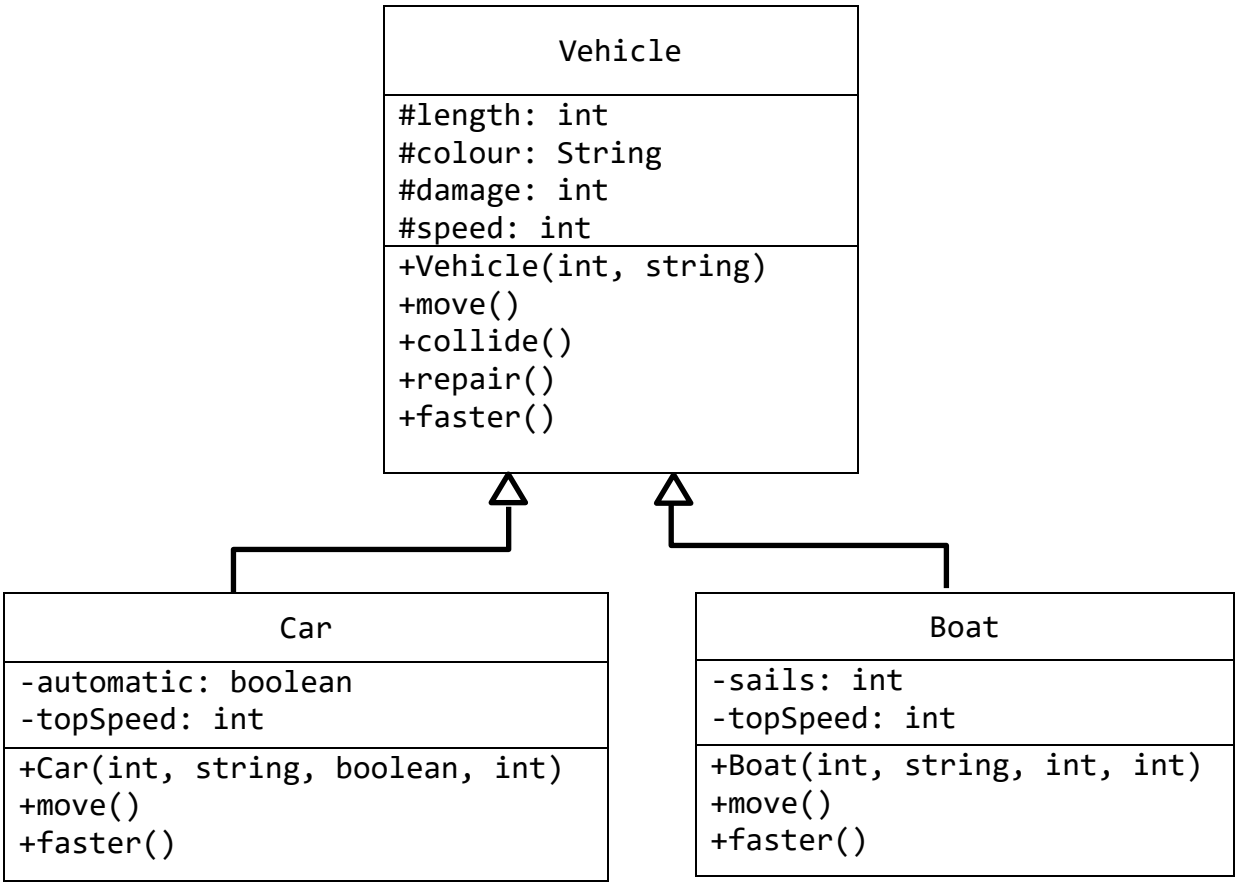
i. Clothing – "shorts", "medium", "khaki", 5

ii. CustomTshirt – "small", "green", "100", "Climate Action Now!"

iii. CustomHoody – "large", "blue", 40

iv. Run `setText` on the object you created in part iii) and give it the values "Cranhurst College CS", "Class of 2022"

v. Run `displayDetails` on all 3 objects to see that you get the output you expect



6) Create a new project and three classes within it as per the diagram above

a. Implement the `Vehicle` constructor so it has a length and colour argument stored into the relevant attribute and initial `speed` and `damage` values of 0

b. A Car can be automatic or manual, implement the constructor so an extra argument determines the value of the `automatic` attribute, and another argument sets the `topSpeed` attribute

c. A `Boat` always has a motor, but can have 0 or more sails. Implement the constructor so that the number of sails is passed in as an argument to set the `sails` attribute

d. Implement the `collide` method so that it adds 10 to `damage` each time it is called up to a limit of 100, and halves `speed` whilst `speed` is more than 0.

e. Implement the `repair` method so that it resets `damage` to zero when called

f. Implement the `faster` method so that:

i. it increases the value of the `currentSpeed` attribute by 1 as long as it is less than 5 for `Vehicle`

ii. it increases the value of `currentSpeed` by 10 so long as it is less than `topSpeed` for `Car` objects

iii. it increases the value of currentSpeed by 2 so long as it is less than `topSpeed` for `Boat` objects

g. Implement the move method in:

i. `Vehicle`, so that it displays a message saying `"Vehicle is moving at "` followed by the speed and `" mph"`, while `damage` is less than 100, otherwise it displays `"Vehicle cannot move"`

ii. `Car`, so that it displays a message saying `"Car is being driven at "` followed by the speed and `" mph"`, while `damage` is less than 100, otherwise it displays `"Car cannot be driven"`

iii. `Boat`, so that it displays a message saying `"Boat is being sailed at "` followed by the speed and `" knots"` while damage is less than 100, otherwise it displays `"Boat cannot be sailed"`

h. Test as follows:

i. Create a `Vehicle`, length 90, colour black. Run `faster` followed by `move` 5 times and confirm that the speed no longer increases

ii. Create a `Car`, length 400, colour silver, automatic, with a top speed of 50. Run `faster` followed by `move` 6 times and confirm that speed no longer increases

iii. Create a `Boat`, length 600, colour white, sails 2, top speed 10. Run `faster` followed by `move` 5 times and confirm that `speed` no longer increases

iv. Run the `collide` method on any of the objects 10 times and follow up with a `move` to confirm that the damage results in a situation where the vehicle cannot move and that the speed has reduced to zero.

v. Run the `repair` method on the same object, and attempt to run `faster` and `move` methods to confirm vehicle can move again

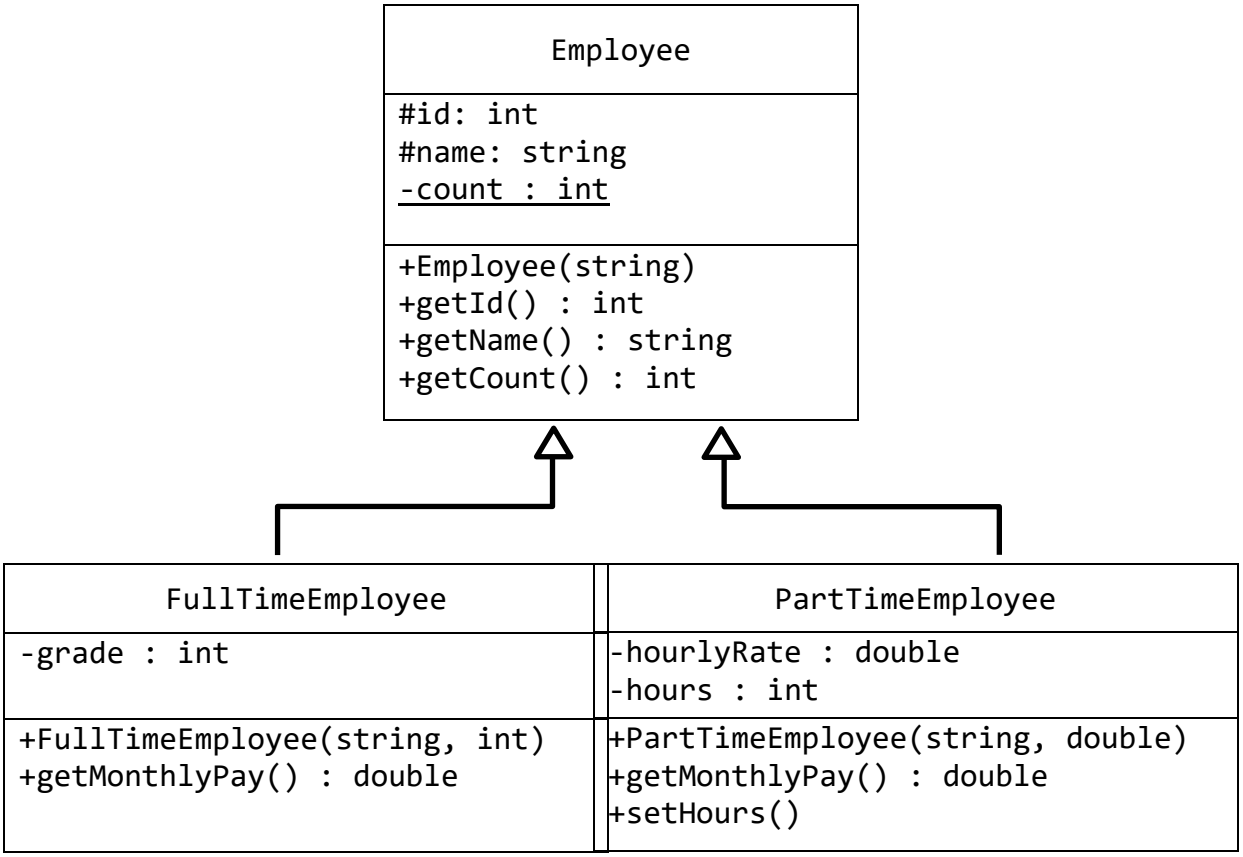
Answer Key

To view the answers to these exercises please refer to the appropriate videos.

Polymorphism

Questions

1) Create a new project and three classes within it as follows:



2)

a. Implement all the selector methods within `Employee`
b. The constructor for `Employee` should store the argument into `name` and use `count` to keep track of how many employees there are in total by incrementing `count` each time the constructor is called, and setting the instance variable `id` to that number (hence ensuring each employee has a different, automatically generated id)
c. The constructor for `FullTimeEmployee` has an extra argument which should be used to set the pay grade
d. The constructor for `PartTimeEmployee` has an extra argument which should be used to set the hourly pay rate
e. `getMonthlyPay` for `FullTimeEmployee` should return a value which is 800 multiplied by the pay grade

f. `getMonthlyPay` for `PartTimeEmployee` should return a value which is `hours` multiplied by `hourlyRate`
g. `setHours` should accept a single `int` argument which is stored into the `hours` instance variable
h. Test as follows on the BlueJ object workbench:
   i. Create an object from the base class with the argument "Bob"
   ii. Create an object from the `FullTimeEmployee` class with the arguments "Stu" and 1
   iii. Create an object from the `FullTimeEmployee` class with the arguments "Petra" and 2
   iv. Create an object from the `PartTimeEmployee` class with the arguments "Jo" and 10.5, then run `setHours` with an argument of 10 on this object
   v. Create an object from the `PartTimeEmployee` class with the arguments "Jinnie" and 11.5, then run `setHours` with an argument of 20 on this object
i. Run `getCount` to see that 5 is returned for number of employees
j. Run `getMonthlyPay` on each respective object, which should return:
   800, 1600, 105 and 230

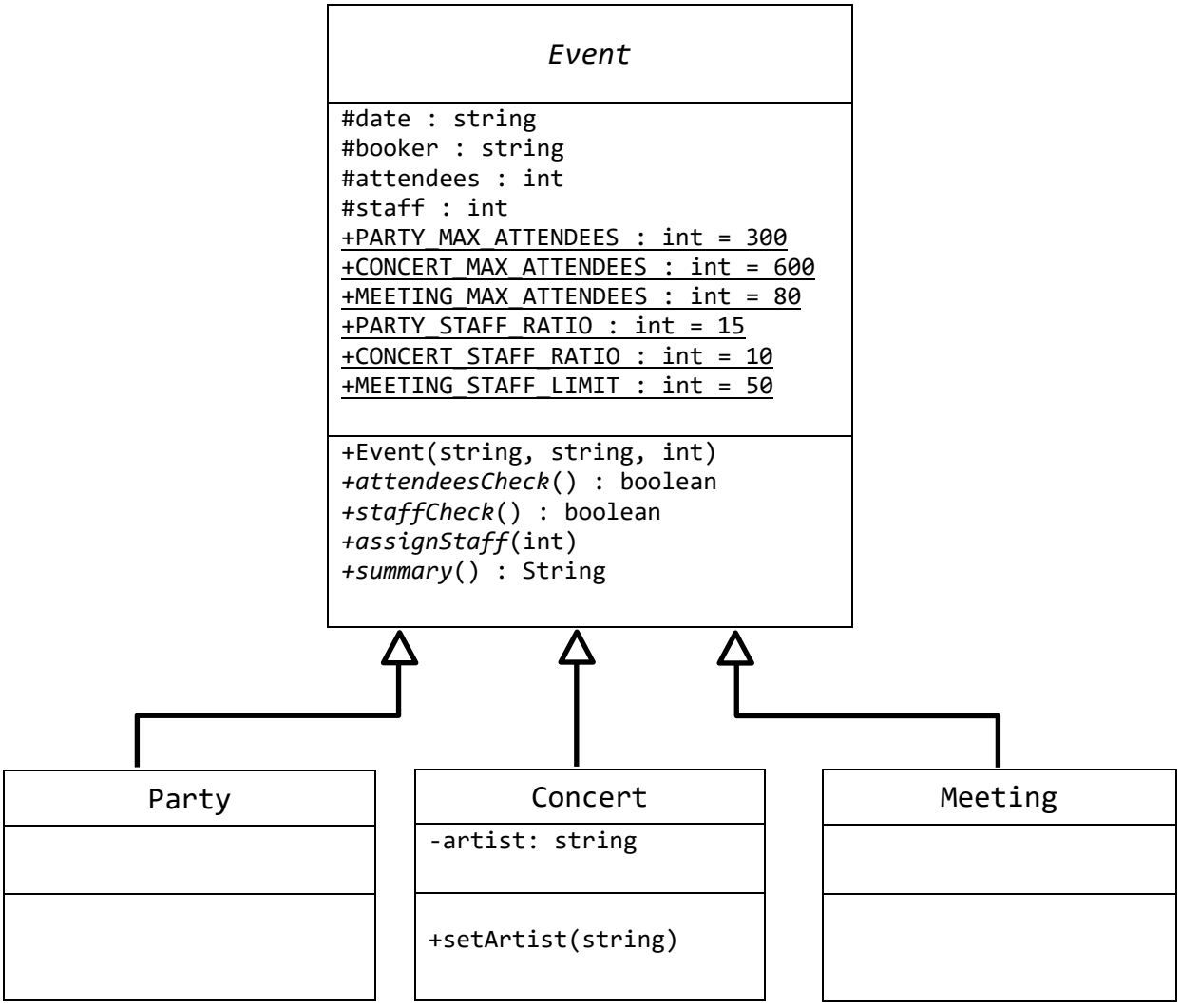
2) Change the `Employee` class from the previous exercise into an abstract class.
a. On the object workbench, once again attempt to create a part time employee, a full time employee and an employee from the base class. What has changed, and why is this?
b. Add the `@Override` annotation before the `getMonthlyPay` method in either the full time or part time employee class. What happens, and why?
c. Implement a dummy `getMonthlyPay` method in the relevant class which always returns the value 0. Add `@Override` annotations before `getMonthlyPay` in the two relevant places and confirm there are no compiler error messages
d. Remove the dummy `getMonthlyPay` method and replace it with an abstract method. Once again confirm there are no compiler error messages. How is an abstract class different to an abstract method?

3) Create a new class `AppLaunch` within the project from the previous exercise

a. Within `AppLaunch` create a public static method `main` which accepts an argument of a String array and returns no values
b. Create three variables inside the `main` method, `emp1`, `emp2` and `emp3` of type `Employee`
c. Instantiate an object of type `PartTimeEmployee` with arguments "Dot" and 10.5, and assign it to the `emp1` variable
d. Instantiate an object of type `FullTimeEmployee` with arguments of "Dash" and 1, and assign it to the `emp2` variable
e. Add two lines of code using `System.out.println` to output the monthly wages using the two objects stored in `emp1` and `emp2`
f. Assign the contents of `emp1` to `emp3` and output the monthly wages of the object stored in `emp3`
g. Assign the contents of `emp2` to `emp3` and output the monthly wages of the object stored in `emp3`
h. Compile the code and run the `main` method, you should see an output of: 0, 800, 0, 800
i. How was it possible for `emp3` to be assigned the value of `emp1` OR `emp2` when they are both different types?
j. Dot's hours are currently zero. Attempt to add a line to run the `setHours` method for `emp1` where Dot is currently stored. Use an argument of 10
k. What compiler error is generated and why?
l. This problem can be fixed by moving an attribute and method in one of the classes to a different location. Implement the changes that will enable the program to successfully compile. Test that running `main` now produces an output of 105, 800

4) A venue holds events and needs a computer program to manage bookings. The venue can cope with a different number of attendees depending on the type of event, and staffing levels for the event also depend on the type of event

a. Create a new project and within it classes as follows (note the italics and capital letters):



3)

b. The constructor for an `Event` requires a date (as a string), the name of the booker, and the number of people who will attend the event

c. Implement the behaviours in the subclasses according to the following criteria:

i. `attendeesCheck` returns `true` if the number of attendees is less than or equal to the maximum number of attendees for that type of event, e.g. for a party, 250 would return `true` but 301 would return `false`

ii. `staffCheck` returns `true` if the number of staff currently assigned is greater than or equal to the number of attendees divided by the staffing ratio for the relevant event, e.g. for a party with 150 people, a value of 10 assigned to staff would return `true`. Note meetings only ever need 1 staff member unless there are more than 50 attendees, in which case there should be 2 staff members. This threshold value of 50 is held in MEETING_STAFF_LIMIT

iii. `assignStaff` sets the `staff` attribute to the correct amount for the type of event being held, e.g. for a concert with 600 people, `staff` would be set to 60

iv. `setArtist` is a modifier which takes a string argument and uses it to set the value of the `artist` attribute

d. The behaviour for `summary` is as follows:

i. For a party, summary returns a string with the booker's name followed by "'s party.", concatenated with the date, concatenated with the number of attendees e.g. "Abigail's party 10/2/2022. 150 invited"

ii. For a concert, summary returns "Concert: " concatenated with the artist, concatenated with the date, concatenated with the number of attendees e.g. "Concert: Codepunks 11/2/2022. 500 capacity crowd"

iii. For a meeting, summary returns booker concatenated with " Meeting " concatenated with the date e.g. "Planning Committee Meeting 12/2/2022"

e. Test your code by creating objects on the BlueJ object workbench as follows:

i. A party with arguments "10/2/22", "Abigail", 150

ii. A concert with arguments "11/2/22", "A. Gent", 500. Use `setArtist` to set the `artist` attribute to "Codepunks"

iii. A meeting with arguments "12/2/22", "Planning Committee", 81

iv. Check the output from `summary()` for each of the objects above matches parts di)-diii)

v. Run `assignStaff` for each object, and inspect each object to check if the correct number of staff are allocated (10, 50 and 2 respectively)

vi. Run `attendeesCheck` for each object, all should return `true` apart from the meeting object

5) This exercise makes use of the classes `Event`, `Party`, `Concert` and `Meeting` created in the previous exercise

a. In the existing project, create a new class called `EventManager`, and within it a standard Java `main` method
b. Inside `main`, declare an array of `Event` objects of size 7, and call it `events`
c. Instantiate three objects and store references to them in the first three locations of the array as follows:
   i. A party with arguments "10/2/22", "Abigail", 150 in `events[0]`
   ii. A concert with arguments "11/2/22", "A. Gent", 500 in `events[1]`
   iii. A meeting with arguments "12/2/22", "Planning Committee", 81 in `events[2]`
d. Use a `while` loop to output the details of each element stored in the array using the `summary` method. The loop ends when a `null` object is encountered, or the end of the array is reached.
e. Test by running the `main` method, and note the issue with the output for the concert. Why has this happened?
f. Attempt to rectify the problem by adding a line after the creation of the concert object. What is the error message you see and why does it occur?
g. Fix the problem by adding a line of code creating a new object called `tempConcert` of type `Concert`. On the same line, assign the current value of `events[1]` to this new object. You will need to use a cast.
h. Use the `tempConcert` object to fix the problem with the missing information spotted in part e)
i. Run `main` again to confirm the program now works as expected

6) This exercise builds on the previous one to build more functionality into the EventManager application and will allow users to book and store events interactively using keyboard input

a. Edit your existing `EventManager` class so it looks like this:

```
public class EventManager{

    Event[] events = new Event[7]; // moved out of main
    int eventPointer=0;

    public static void main(String[] args){
        EventManager app = new EventManager();
        app.menu(); // method not defined yet
    }

    public void listEvents(){ // moved out of main
        int i=0;
        System.out.println("Events:\n");
        while( i < events.length && events[i] != null){
            System.out.println(events[i].summary());
            i++;
        }
    }
}
```

b. Create a new method inside the `EventManager` class called `displayOptions` which returns no values and accepts no arguments. Within it, print out the options below:
   i. '1' – list booked events
   ii. '2' – book a new event
   iii. '3' – check attendance
   iv. '9' – exit program

c. Create another new method called `menu`, which does the following:
   i. Create an object from the `Scanner` class (add `import java.util.Scanner;` at the top of the class) by adding as the first line of the method:
   `Scanner keyb = new Scanner(System.in);`
   ii. Call `displayOptions()` to display the initial options
   iii. Use a `while` loop to take input from the keyboard using `keyb.next()` to either quit the loop or run the `listEvents` method
   iv. Use equals not `==` and `!=` when comparing strings in Java

v. Call `displayOptions()` inside the loop to display the options each time, and then check again for keyboard input

d. Run the `main` method in `EventManager` to see if the code so far works as intended by choosing option 1 and then option 9

e. Implement a selector method in the base class which returns the date attribute

f. Adapt the constructor in the `Concert` class so it accepts an extra argument to set the value of the `artist` attribute

g. Implement a new method `bookEvent` which asks the user for all the data required for an event, and adds it to the next available slot in the array

i. Declare four `String` variables: `date`, `booker, type` and `artist`, and one `int attendees`

ii. Print a prompt for each item of input e.g. "Enter the date:"

iii. Use a Scanner object and next() to get a string from the keyboard, or `nextInt` to get an integer. Also set the delimiter for Scanner to newline, e.g.

```
Scanner keyboard = new Scanner(System.in);
keyboard.useDelimiter("\n");
String input = keyboard.next();
int someOtherInput = keyboard.nextInt();
```

iv. The input for `type` determines what sort of object should be created and stored in the array and can only be "party", "concert" or "meeting"

v. For a type of "concert" you need to take input for the artist name as well as all the other inputs

vi. Use `eventPointer` as an index for the array

h. Test by adding an event of each type interactively using menu option 2 and then run menu option 1 to see the result

i. Implement a new method called `dateFree`() which accepts a string argument and returns a `boolean`. This method should loop through all the elements in the array checking the `date` attribute of each event does not equal the argument passed in, if no matching date is found it returns `true`, otherwise it returns `false`

j. Adapt the `bookEvent` method so that it only allows a booking to be made if the proposed date is free, or otherwise displays a message saying "Date not available"

k. Implement the check attendance menu option to iterate through all the events checking the attendees do not exceed the limit for the event type. If so a warning message is printed followed by the details of the event. Output " Attendance checks completed" before the code exits

l. Test the with the following event data entered:
   i. A party with arguments "10/2/22", "Abigail", 150
   ii. A concert with arguments "11/2/22", "A. Gent", 500, "Codepunks"
   iii. A meeting with arguments "12/2/22", "Planning Committee",81
   iv. A meeting with arguments "12/2/22", "Clash", 20
   (should fail because of date)
   v. Run the check attendees menu option which should highlight the third event as exceeding the attendee limit for meetings

Answer Key

To view the answers to these exercises please refer to the appropriate videos

Collection Classes

Questions

1) Create a new project and within it a class called `CollectionsApp`

a. Within a new method `politeNumbers` which returns no values and accepts no arguments, declare an `ArrayList` called `myList` to contain integers

b. Use the `add` method to add these numbers one at a time to `myList` :

i. 3, 5, 6 – use the `new` keyword

ii. 6, 7, 9 – use autoboxing

c. Output the contents of `myList` using a single `println` statement. Use the object workbench to test that the numbers you expect are in `myList`

d. Using the methods `size()` and `get()`, store the last element of `myList` into a new `int` variable `lastInt`

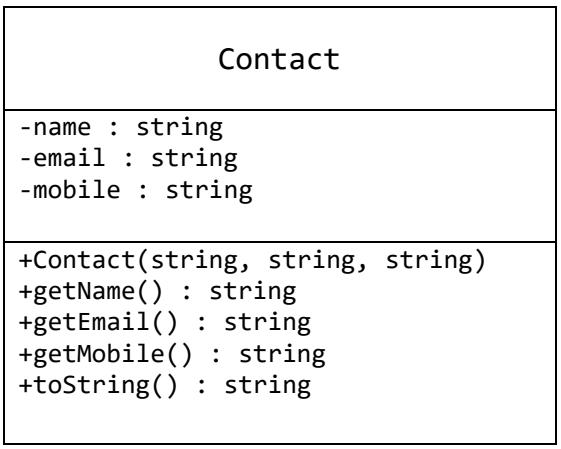
e. Output the contents of the `lastInt` variable on the next line. Use the object workbench again to test that the output is what you expect.

f. Remove one of the occurrences of 6 from `myList` using the `remove` method and output using a single `println` again

g. Test by creating an object on the object workbench and running the `politeNumbers` method

h. Double click on the object you created on the workbench so you can inspect its contents. Why can you not see `myList`? Can you move `myList` so it can be seen on the object workbench?

2) Create a new class within your existing project called `Contact`



a. Implement the attributes, constructor and accessor methods as above
b. The `toString` method overrides the `toString` method inherited from the `Object` class that all Java classes inherit, and should be used to return the attribute values as a single string e.g. "Name: Bill ; Email: bill@tepid.com; Mobile: 07500 123456"
c. Now create a new public method `addressBook`, within your `CollectionsApp` class, which returns no values and takes no arguments.
d. Create an `ArrayList` called `contacts` to hold an arbitrary number of contacts. Ensure this is declared outside the `addressBook` method so it can be inspected on the object workbench
e. Within `addressBook` put 3 contacts into the `contacts` array list using the add method as follows:
   i. Bill, bill@tepid.com, 07500 123456
   ii. Bob, bob@whizzmail.com, 07500 888888
   iii. Bea, bea@zcloud.net, 07500 999999
f. Use a for-each style `for` loop to output each contact in `contacts` on a new line
g. Run the `addressBook` method and see if the contacts are added to the array list and output to the terminal window

3) Modify the `addressBook` method you created in the previous exercise to allow a user to input new contacts and search for them

a. Move the code to display all the contacts into a new method called `listContacts`

b. Use a `Scanner` object and `do...while` loop to take a command from the user:

i. "add" allows a user to enter the details for a new contact and adds them to the contacts list

ii. "list" displays all current contacts

iii. "exit" ends the program

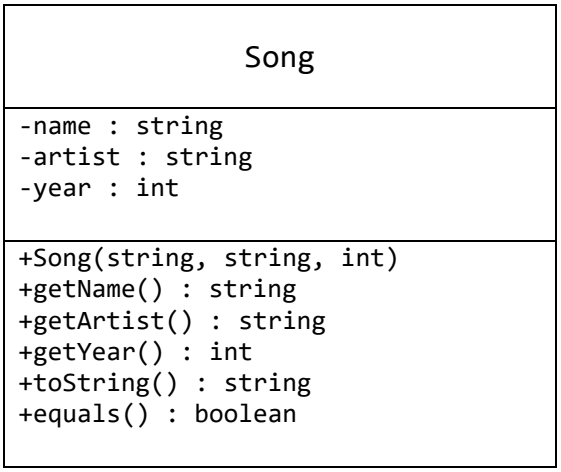
c. Run the program and add two contacts, followed by a list command and exit command

d. Develop the program so a new command of "find" allows a user to find a contact, by entering their name, and either display that contact's details or "Contact not found." Use an Iterator object to iterate through the contacts rather than a for loop

e. Test by adding new contacts "Anita" and "Andrew" and attempt to find Anita's details in the contacts list using the find command

4) A hash code is an integer value that is generated using an algorithm applied to some input. With objects it can be used to identify an object. When a user defined class overrides the `equals` method, it should also override the `hashCode` method, so that objects that are equal produce identical `hashCode` values

a. Create a new project and within it a new class called `Song` as follows:



b. Implement all the methods, including an overridden `toString` method that returns the three fields separated by commas and a single space, e.g. "Yesterday, Beatles, 1965"

c. The overridden `equals` method should compare the attribute values for `name` and `artist`, and should return `true` only if both match in the object being compared

d. Create three objects from this new class on the object workbench using the following data:
   i. Yesterday, Beatles, 1965
   ii. Yesterday, Leona Lewis, 2007
   iii. Yesterday, Beatles, 1965

e. Run tests by creating objects with the same values as before in part d, then check that:
   i. song1 and song3 are equal
   ii. song1 and song2 are not equal
   iii. song2 and song3 are not equal

f. Run the `hashCode` method on each of the objects. Is the integer value returned the same for any of the three objects? Why is this so?

5) Create a new class called `HallOfFame` which makes use of the `Song` class to store songs that have been inducted into the Music Industry Hall of Fame

a. Within it, create a `HashSet` collection of `Song` objects called `songSet`

b. Create a method that returns no values and accepts no arguments called `testSet` and within it add the following data to `songSet`:

i. Yesterday, Beatles, 1965
ii. Many Rivers to Cross, Jimmy Cliff, 1969
iii. You've Got a Friend, James Taylor, 1971
iv. Hotel California, Eagles, 1977
v. Sweet Dreams, Eurythmics, 1983
vi. Walk This Way, Run DMC, 1986
vii. Yesterday, Beatles, 1965

c. At the end of the method print out `songSet` on one line. Sets are not supposed to contain duplicate values, are there any duplicate songs? Why has this happened?

d. Override the `hashCode` method of the `Song` class so that it adds together the `hashCode` value of the `name` attribute and the `hashCode` value of the `artist` attribute. Run the code again and compare the output to what you saw in part c.

6) A `HashMap` can be used to efficiently look up information in a collection of objects by using a key/value pair. Items are put into the `HashMap` using the `put` method, which requires a key and the item to be stored. Values are retrieved using the `get` method with a key, and the `HashMap` can be searched for a particular key using the `containsKey` method

a. Within the existing `HallOfFame` class, create a new collection called `songLibrary` which can store a collection of `Song` objects, indexed using a `String` key

b. We will use a music genre as the key to each of our songs. Within a new method called `testMap`, add the following data to the `songLookup` collection:

i. Key: Rock, Song: Yesterday, Beatles, 1965
ii. Key: Reggae, Song: Many Rivers to Cross, Jimmy Cliff, 1969
iii. Key: Folk, Song: You've Got a Friend, James Taylor, 1971
iv. Key: Country Rock, Song: Hotel California, Eagles, 1977
v. Key: Electronic Pop, Song: Sweet Dreams, Eurythmics, 1983
vi. Key Hip-Hop, Song: Walk This Way, Run DMC, 1986

c. Test by outputting the contents of the `songLookup` collection on a single line

d. Use a `while` loop to take input from the keyboard, prompting the user to enter a genre or the keyword "quit" to end. If a song from that genre is found in the `HashMap`, output the details, otherwise output the message "Genre not found"

e. Test your code works by searching for some genres you know to be present in the hash map, and with at least one other that you know is not

f. What is the flaw in using the genre as a key in this particular exercise, and how might you solve it?

Answer Key

To view the answers to these exercises please refer to the appropriate videos

Design Principles

Questions

1) Create a new project called `Aggregation`

a. Create a new class called `Page` as follows:
   i. Add two attributes, a `String` called `content` and an `int` called `pageNumber`
   ii. Create a constructor which sets the two attributes to the values passed in
   iii. Create an overridden method `toString` which returns a newline character, concatenated with the `content` attribute, concatenated with another newline, and then the `pageNumber` attribute, followed by one final newline character

b. Test your `Page` class on the BlueJ object workbench, by creating an object, with the content attribute set to "Test page" and page number as 1, and then run the `toString` method to see the output includes the content and page number

c. Create a new class called `Book`, as follows:
   i. Add a single `String` attribute called `title`, and a constructor which sets its value
   ii. Add an `ArrayList` called `pages` which stores a collection of `Page` objects
   iii. Add a method called `addPage` which accepts a `Page` object as a parameter, and then uses the `add` method of the array list `pages` to add the passed in page to the book
   iv. Add a method called `readBook` which first outputs the title of the book followed by a newline character, and then the entire contents of the `pages` collection using a loop

d. Test on the object workbench by first creating pages with content:
   i. "I remember him as if it were yesterday, as he came plodding to the inn door..." (page 1)
   ii. "He was a very silent man by custom. All day he hung round the cove or upon the cliffs with a brass telescope." (page 2)
   iii. "His stories were what frightened people worst of all." (page 3)

e. Still on the object workbench, create a Book object with the title "Treasure Island"

f. Use the addPage method of the book object, to add each page you created in part d) to the book in the correct order, i.e. if your first page object was

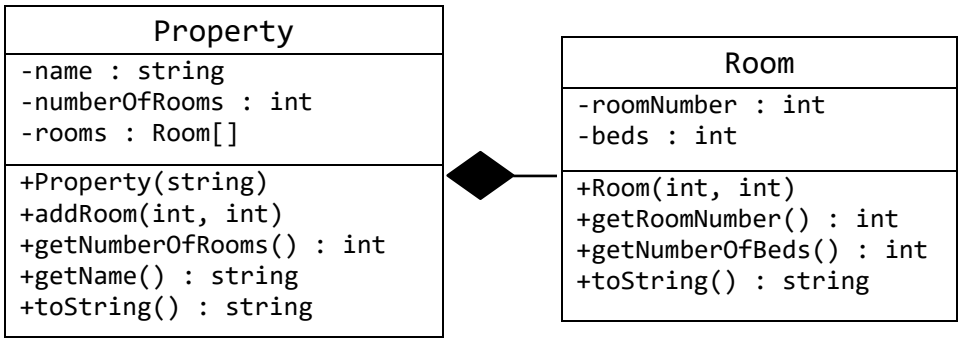
called page1, add that first to the book using addPage,
then page2 and so on.

g. Run the `readBook` method to see whether the output in the terminal window matches what you expect for a 3 page book
h. What sort of relationship is there between the book and page? (ignore the arrow drawn by BlueJ)

2) Remaining in your existing project in BlueJ

a. Type the following into the Code Pad and note the effect:
   i. import java.time.*;
   ii. LocalDate date1 = LocalDate.of(2022,05,21);
   iii. LocalDate date2 = LocalDate.of(2022,05,21);
   iv. LocalDate date3 = LocalDate.of(2022,05,33);
   v. LocalDate date3 = LocalDate.of(2022,05,31);
   vi. LocalDate date4 = LocalDate.now();
   vii. date1.equals(date2)
   viii. date1.equals(date3)
   ix. date1.getMonth()
   x. date3.getDayOfMonth()
   xi. date4.getDayOfWeek()
   xii. date3.isAfter(date1)
   xiii. date1.isBefore(date4)
   xiv. date1.isLeapYear()
   xv. date4.toString()
b. Which particular OO design principle does the LocalDate class exhibit?

3) Create a new project called `Hotels` and two classes within it as follows:



a. A property contains rooms, each of which has a room number and an indication of how many beds the room has. Add the attributes and the constructor for `Room` to set the two attributes

b. After adding the selector methods, override the `toString` method for `Room` to return a string that concatenates the room number and number of beds, e.g. "Room 11 has 1 bed(s)"
c. The `Property` class should use an Array List collection called `rooms` to store its rooms
d. Rooms are added to objects created from the `Property` class using the `addRoom` method, which:
   i. creates an object from the `Room` class with the first argument used to set the room number, the second to set the number of beds
   ii. adds the created room to the collection called `rooms`
   iii. updates the `numberOfRooms` attribute to correctly reflect the number of rooms at the property
e. After adding the selector methods, override the `toString` method for this class to return a string that concatenates the property name and number of rooms in brackets, e.g. "Regent Hotel (5 rooms)"
f. Create another class called `HotelsApp` which we will use to test our `Room` and `Property` classes
g. Inside this class declare a `HashMap` called `properties` which has a `String` key and is used to store `Property` objects
h. Within the `HotelsApp` class, create a method called `start`, which will be used to start our application. Inside `start`, create an instance of a `Property`, with a name of "Regent Hotel" and store a reference to the object in a variable called `hotel`
i. Using the `addRoom` method, add rooms to the object referenced by `hotel` as follows:
   i. Room 1, with 1 bed
   ii. Room 2, with 1 bed
   iii. Room 3, with 1 bed
   iv. Room 11, with 2 beds
   v. Room 12, with 2 beds
j. Add the hotel object just created, to the `properties` collection using `put`, with the key being the name of the hotel
k. Create a new instance of a property, this time with a name of "Crummy Inn" and store a reference to this new object in the variable `hotel` which has already been declared. Now again add rooms to the object referenced by `hotel` as follows:
   i. Room 1, with 1 bed
   ii. Room 2, with 2 beds
   iii. Room 3, with 3 beds

l. Again, add the hotel object just created, to the `properties` collection using `put`, with the key being the name of the hotel
m. Test by adding a line to print `properties` and run the `start` method on the object workbench
n. Can you explain the output you see generated?

4) We will now build on the project created in the previous exercise to implement a hotel booking system. Create a new enum by clicking on the "New Class" button on the left hand side of the project screen and in the dialog that comes up, selecting "Enum" under "Class Type". Give your enum the name `Command`

a. Edit the default code generated by BlueJ so that the values inside the braces are: `QUIT, BOOK, LIST, MENU, BOOKINGS`
b. Compile the enum, fixing any errors as appropriate
c. Reopen the `HotelsApp` class in the project, and above the `start` method add another HashMap, called `commandWords`, which uses a `String` as a key, and a `Command` enumerated type as the value to be retrieved from the collection
d. Inside the `start` method add a series of entries to the HashMap as follows:
   i. Key: "quit", Command: Command.QUIT
   ii. Key: "book", Command: Command.BOOK
   iii. Key: "list", Command: Command.LIST
   iv. Key: "menu", Command: Command.HELP
   v. Key: "exit", Command: Command.QUIT

e. Test that the map is correctly setup by adding the line `System.out.println(commandWords.keySet());` and running the `start` method from the object workbench
f. Add the following code within start below the code written so far:

```
Command command=Command.HELP;
while(command!=Command.QUIT){
     processOption(command);
     command = chooseOption();
}
```

g. Inside the HotelsApp class, implement the method processOption which accepts a single argument of type `Command`
   i. Use a `switch` structure with a `case` for each of the enumerated commands, for example the `case` for `HELP` should be

```
case HELP :
   System.out.println("Valid command words are the following:"
+ commandWords.keySet());
   break;
```

ii. Implement the code for the LIST command so that it returns a list of the available properties stored in the `properties` HashMap
iii. Add a default case that simply displays a message saying "Command not yet implemented"

h. Implement the method `chooseOption` as follows:
i. Create an object from the `Scanner` class and store a reference to it in a variable `keyboard`, then a String variable called `choice` to contain user input and a `boolean` variable `valid` used as a flag variable initially set to `false`
ii. Create a `while` loop which continues looping while `valid` is false. The inside of the loop should take input from the keyboard, and check whether the input string is a valid key in the `commandWords` hash map by attempting to retrieve the value using `get`. If `get` returns `null` a message is displayed saying "Invalid command. Try again." otherwise `valid` should be set to `true` to enable the loop to terminate
iii. Outside the while loop, the method should return the value of the command retrieved from the hash map

i. Test by running the `start` method and typing the following commands:
i. list (should return a list of the hotels)
ii. help (should show the command list)
iii. book (should show "Command not yet implemented")
iv. make(should force you to type in another command)
v. exit or quit (should exit the program)

j. Why was the code we added in this exercise broken up into so many individual pieces rather than just written out as a single block inside start?

5) We will again build on the previous exercise, this time to allow a room booking to be made at any of the properties available

a. Create a new class, `Booking`, with three private attributes:
   i. A reference to a `LocalDate` object called `date`
   ii. An `int` called `beds`
   iii. A reference to a `Property` object, called `property`

b. Add a constructor method to the class that sets the three attributes

c. Override the `toString` method within `Booking` so that it concatenates "Date: " with the date, a comma, the property name, a comma, and the number of beds e.g. "Date: 2022-05-22, Regent Hotel, 2 bed(s)"

d. Add an ArrayList collection inside the `HotelsApp` class called `bookings`, which will hold objects of type `Booking`

e. Test by adding the following lines of code inside the `start` method, immediately below the creation of the first property (Regent Hotel):
   i. `Booking booking = new Booking(LocalDate.of(2022,06,22),1,hotel);`
   ii. `bookings.add(booking);`
   iii. `System.out.println(bookings);`
   iv. Run the `start` method and check the booking is displayed

f. Create a new method `inputProperty` which has no arguments and returns a `Property` object reference. It should:
   i. prompt the user to enter a property name at the keyboard
   ii. store the property entered at the keyboard into a string (allow for a property name that includes spaces)
   iii. check whether that property exists in the `properties` collection
   iv. if it does not, display a message "Not found, try again." and loop back to i) above
   v. when a valid property is entered, the method should return a reference to that property

g. Test the `inputProperty` method by adding this line below the booking test code within `start` as follows and running the `start` method:
   `System.out.println( "*Found* " + inputProperty() );`
   i. Check an input of "Notel" is rejected and prompts for more input
   ii. Check an input of "Regent Hotel" is accepted and returns a `Property` object so the found message is displayed along with the hotel details

h. Add a similar method, `inputDate` which has no arguments but returns a reference to an object of type `LocalDate` as follows:
   i. Prompt the user to type in an integer from 1 to 31 and store the input from the user into an integer variable `day`

   ii. Prompt the user to type in an integer from 1 to 12 and store the input from the user into an integer variable `month`
   iii. Prompt the user to type in an integer representing the year and store the input from the user into an integer variable `year`
   iv. Use a `try…catch` block to attempt to create a `LocalDate` object using the values typed in and the `of` method in `LocalDate`
   v. If the date created is on or before today's date - as obtained by `LocalDate.now()` – display a suitable message and force the user to type in another date
   vi. In the catch block display the error message "Date is not valid."
   vii. otherwise return a reference to the newly created `LocalDate` object

i. Test the `chooseDate` method by creating an object from `HotelsApp` on the object workbench, as follows:
   i. Check a date of 31-06-22 is rejected and invalid date message is displayed
   ii. Check a date in the past is rejected and a suitable message is displayed
   iii. Check a valid date after today is accepted and returns a `LocalDate` object

j. Remove any debug code within the start method from previous exercises
k. Add a new method called `makeBooking` with no arguments or return values which calls the `inputProperty` and `inputDate` methods, and then using the inputted data, adds a booking to the `bookings` collection. Assume the booking will always be for a room with 1 bed.
l. Add another method called `listBookings` which again returns no values and takes no arguments, but whose implementation iterates through the bookings collection, and outputs each booking on a new line
m. Add in the necessary code to implement "book" and "bookings" commands which respectively enable booking and listing of current bookings
   i. Make sure the command keywords are in the `commandWords` collection
   ii. Make sure the commands all have a `case` in `processCommands`
   iii. Make sure the commands are listed in the `Command` enum
n. Test by adding at least 2 valid bookings for each of the hotels, and then run "bookings" each time to see the impact

6) Considering the Hotel application built in the previous exercises, answer the following:

a. The core of the program is in the `start` method. It currently violates one of the design principles of well written OO programs, how could it be improved?

b. As it stands, when making bookings, the program doesn't check whether there is availability on the particular date the booking is made. How and where can that be addressed?

c. An important entity is missing from the design for this application, what is it?

d. Check in involves allocating a room to a guest who has made a booking and arrives at the hotel. The room number is given to them based on the type of room they booked (how many beds) and which rooms are available at the time they check in. How might this check-in routine work?

Answer Key

To view the answers to these exercises please refer to the appropriate videos