COMP1549 (2018/19)	**Advanced Programming** Practical Coursework	**Header ID** 300663	**Weighting** 50%
Course Leader Zena Wood	**Release Date** 8th February 2019		**Deadline Date** 9th April 2019

This coursework should take an average student who is up-to-date with tutorial work approximately 25 hours.

Feedback and grades are normally made available within 15 working days of the coursework deadline.

Learning Outcomes:

1. Use professional techniques for code and design reuse such as library creation, application of design patterns and development of software components.
2. Apply advanced programming techniques such as threads, reflection and generic classes, basic distributed programming techniques, object relational mapping.
3. Make appropriate use of software engineering tools and techniques such as test-driven development, version control, generation of documentation, build-tools.

Plagiarism is presenting somebody else's work as your own. It includes: copying information directly from the Web or books without referencing the material; submitting joint coursework as an individual effort; copying another student's coursework; stealing coursework from another student and submitting it as your own work. Suspected plagiarism will be investigated and if found to have occurred will be dealt with according to the procedures set down by the University. Please see your student handbook for further details of what is / isn't plagiarism. Details are also on the Student Hub on Moodle.

All material copied or amended from any source (e.g. internet, books) must be referenced correctly according to the reference style you are using. Code snippets from open source resources or YouTube must be acknowledged appropriately.

Your work will be submitted for electronic plagiarism checking. Any attempt to bypass our plagiarism detection systems will be treated as a severe Assessment Offence.

Coursework Submission Requirements

- An **electronic copy** of your work for this coursework must be fully **uploaded by 23:55 on 9th April 2019.**
  - Submissions must be made using the Coursework link under "Coursework Specification and Upload" on the Moodle page for COMP1549.
  - Your **work will be capped if you fail to submit by the deadline**.
- For this coursework you must submit a **zip file containing all the files required to run your java project and a single PDF document containing your report**.
- In general, any text in the document must not be an image (i.e., must not be scanned) and would normally be generated from other documents (e.g., MS Office using "Save As .. PDF"). There are limits on the file size.
- Make sure that any files you upload are virus-free and not protected by a password or corrupted, otherwise they will be treated as null submissions.
- Your work will be marked online and comments on your work and a provisional grade will be available from the Coursework page on Moodle. A news item will be posted when the comments are available, and also when the grade is available in BannerWeb.
- All coursework must be submitted as above. Under no circumstances can they be accepted by academic staff.

The University website has details of the current Coursework Regulations, including details of penalties for late submission, procedures for Extenuating Circumstances, and penalties for Assessment Offences. See http://www2.gre.ac.uk/current-students/regs

Coursework Regulations

- If you have Extenuating Circumstances you may submit your coursework up to two weeks after the published deadline without penalty, but this is subject to your claim being accepted by the Faculty Extenuating Circumstances Panel.
- Late submissions will be dealt with in accordance with University Regulations.
- Coursework submitted more than two weeks late may be given feedback but will be recorded as a non-submission regardless of any extenuating circumstances.
- Do not ask lecturers for extensions to published deadlines - **they are not authorised to award an extension**.

Please refer to the University Portal for further detail regarding the University Academic Regulations concerning Extenuating Circumstances claims.

Detailed Specification

Please read the entire coursework specification before starting work. This coursework comprises two parts: part A and part B. To achieve full marks you will need to complete both parts based on the specifications below.

Development for both parts A and B is expected to be carried out using pair programming. Using pair programming means that both members of the team should have an equal understanding of the code produced and will be expected to be equally able to explain it during the acceptance tests (demonstration). Further details on pair programming can be found in the lecture notes for week 2 on Moodle.

Decide who you will work with and register your pair via an email to Dr. Zena Wood by 15th February 2019 at the latest (ONLY ONE PARTNER from each pair needs to register). If you are finding it difficult to find a partner, please inform your tutors and we will pair you up with someone in your tutor group.

Part A – Logbook (12% of coursework mark)

The logbook is a compilation of the work produced for the exercises listed in the table below. **Each exercise can be found at the end of the tutorial exercise slides for the specified tutorial and will be clearly identified there as a *Logbook* exercise.** These exercises will help with tasks in part B. They are not the same as part B tasks, but will help you with the concepts and skills needed for part B. You are encouraged to complete the tasks required for each exercise as soon as possible after the exercise is announced and show your work to your lab tutor during the following lab session for formative feedback.

Complete the logbook template provided on the COMP1549 Moodle page for each exercise. You may complete these exercises with another person using pair programming. You must state the name of your partner in the template document. **Include a copy of each of the completed template documents in section 8 of your final report as described in the deliverables section below**.

Description	Week	Related level in part B
Logbook 1 – Inheritance Refactor supplied code to make appropriate use of inheritance.	Wk1 18/01/2019 "Java Topic Review"	Level 2
Logbook 2 – JUnit testing Get supplied code from Git server, add JUnit testing to specified class, push back to Git server.	Wk3 01/02/2019 "JUnit" lecture	Levels 1 and 3
Logbook 3 – Design Patterns Refactor supplied code to use the Factory pattern.	Wk4 08/02/2019 "Design Patterns"	Level 4
Logbook 4 – "Creating Software Components" Refactor supplied code to create a JavaBean component.	Wk7 22/02/2019 "Software Components"	Level 5

Part B – Software Development (88% of coursework mark)

We are all familiar with vehicle dashboards and control panels for cars, motorcycle, aircraft, ships etc. (https://www.google.co.uk/search?q=vehicle+dashboards&tbm=isch). Dashboards are also used in a lot of other areas to provide people with summarised information about processes and data (e.g. business, manufacturing, networks, healthcare etc.) see http://www.dundas.com/gallery/sample-dashboards/ for some examples.

You are working for a company that is planning to start development of dashboard applications using Java. The company wants to explore technologies and design approaches that they may use in their dashboard applications. Their aim is to improve their ability to design software that it is **easy to maintain** and **easy to extend**. They are also interested in the concept of "**reuse**". In order to explore and illustrate these concepts they want you to do some work on developing a prototype dashboard application. In particular they want you to demonstrate: good object-oriented design, design patterns, component creation, and use of threads. They would like your application to be a simple simulation of a dashboard for either a train or small aircraft. You choose which.

You are expected to carry out the development using pair programming. If you choose to work alone you will need to justify your reasons. Using pair programming means that both members of the team should have an equal understanding of the code produced and will be expected to be equally able to explain it during the acceptance tests (demonstration).

You will be supplied some initial code. This can be found under Coursework Specification and Upload on the Moodle page for this course. You may use and adapt this as you wish but you should make all updates clear.

You are required to use Git for version control throughout your software development. You should make frequent commits of your work to your local Git repository and regularly push your work to a remote Git repository that will be made available to you. You will be given details about how to do this.

Although graphics is an important part of dashboard design, it is NOT an important part of what is being evaluated in this case. For this reason you should keep the graphics simple and/or may make use of code you find on the web or in books that helps with the implementation specifically of the graphical elements of your application. You must clearly identify this code as code comments and in your report. The rest of the code, including all the logic, should be your own work.

The company would like you to proceed by as follows. The requirements are expressed as a set of levels. The level that you achieve will have a significant effect on your mark for this work. The % shown next to each level is a maximum. You may receive less than this depending on the quality of your implementation and report. Note that the size of your application in terms of numbers of lines of code and number of types of controls implemented is not as significant to your mark as the quality of the code and design features you implement. You are advised to develop your code a level at a time, but you should read through all the levels before starting as the way you develop the earlier levels may be influenced by the later levels you plan to attempt.

Level 1 – Basic application (up to 45% of part B)

Implement a prototype dashboard for a train or small aircraft – you choose which. You may adapt or expand the code given to you by your tutor or start from scratch.

The application should at a minimum have the following:

a) A vertical bar indicator and at least three other display indicators of at least two different types (see appendix A for ideas about types). For example:

- one vertical bar plus two dials plus one digital display
- or one vertical bar plus one traffic light plus one half dial plus one dial

- or any other combination of your choosing.

b) It must be possible to change the data displayed by the indicators both by user input and via a script loaded from a file. The code given to you will demonstrate simple ways of doing this which you may adapt if you wish.

Aim to make your application more reliable, flexible and easy to use than the example code you are given.

Note that the design choices made and the variety of ways data can be changed mean it is very unlikely that two people will independently come up with similar code. Please be careful to do your own work and not copy from your fellow students.

Level 2 – Object-oriented design features (up to 55% of part B)

This level builds on what you did for level 1. The aim is for you to demonstrate how good use of object-oriented features can improve the design of your system in order to make it easier to maintain and enhance in the future. In addition to level 1, this level should:

a) Have at least one additional display indicator of a different type (so in total that is at least five display indicators of at least four different types).

b) Make use of the following object-oriented features

i. **Inheritance**. Create **at least one appropriate inheritance hierarchy** consisting of **at least three classes**. All three classes should be ones declared in the program and not classes from the Java API. Note that inheritance can help to reduce duplicate code and can enhance flexibility by declaring variables, parameters and return types to be of the superclass type. These are the sorts of benefits you should try to demonstrate.

ii. **Interfaces**. Create and make appropriate use of **at least two interfaces**. Note that interfaces can enhance flexibility by declaring variables, parameters and return types to be of the interface type. This is the sort of benefit you should try to demonstrate.

There are a large number of different ways this can be implemented so it will be very surprising if two people independently come up with similar code.

Level 3 – JUnit Testing (up to 60% of part B)

This level builds on what you did for level 2. The aim is for you to demonstrate techniques for carrying out good unit testing using JUnit.

Pick one or more classes that it is useful to test using JUnit. **Implement thorough JUnit testing**. Note that this will normally mean creating a number of test cases for each method of the class being tested.

Level 4 – Design Patterns (up to 70% of part B)

This level should incorporate the features of level 3. In addition, this level should include:

a) an appropriate implementation of the Singleton design pattern and

b) at least two other design patterns. At least one of the design patterns (apart from the Singleton) should be from those studied as part of the course. At least one design pattern should be one not studied as part of this course but researched yourself.

Aim to demonstrate how good use of design patterns can improve the design of the system in order to make it easier to

maintain and enhance in the future. As with the other levels there are lots of different ways this can be done. Do your own work so as not to put yourself or your friends in danger of being accused of plagiarism.

Level 5 – Componentisation (up to 80% of part B)

This level should incorporate the features of level 3. In addition you should create one or two reusable JavaBean components and demonstrate their use in your application. Each JavaBean should:

- adhere to the standard for JavaBeans as covered on the course,
- have **at least four useful properties** (i.e. matching get and set methods),
- be able to be added to the NetBeans component palette and dragged and dropped onto a JFrame as part of another very simple application,
- have an appropriate BeanInfo file which provides an icon for the bean which appears on the palette.

Aim to develop flexible components that could be genuinely useful to other programmers and demonstrate how componentisation makes code easy to reuse code within an application and in other applications.

Level 6 – stretch target (up to 100%)

Only attempt this if you feel that you have done a very good job for levels 1 to 5 and want an additional challenge. This level should incorporate the features of level 5. In addition demonstrate the appropriate use of threads in implementing one or both of the following.

a) In the example code you were supplied, the movement of the display indicators is jerky. Make it so that in your application the indicators move smoothly from one position to another.

b) Implement a simple game based on the dashboard. For example players could attempt to travel a certain distance before fuel runs out or within a certain time.

Deliverables

Part A

1. Upload a completed logbook template for each the four exercises specified.
2. **Include a copy of each of the completed template documents in section 8 of your final report**.

Part B

1. A **zip file** which itself contains one or more zipped NetBeans projects.

    - As each level builds on earlier levels you only need to include the project for the highest level you have reached. For example, if you have achieved level 4 then just include your level 4 project, there is no need to include the level 1, 2 or level 3 projects unless your feel there is a particular need for you to do so. Please make sure to include any necessary script files. If you have completed level 5 then **in addition to your main project please include one project per JavaBean which you have created**.
    - Please structure your work so that it is easy to identify what you have achieved. It should be possible to copy your projects to a PC and run them from there without having to make changes to the code. Any compilation, installation or running instructions should be included in a readme.txt file.
    - If you have borrowed code or ideas from elsewhere (e.g. from a book, some resource on the web or other students) then include a reference showing where the code or ideas came from and annotate your code

carefully to show which bits are yours and which bits are borrowed.

2. Acceptance test (demonstration) of what you have achieved. You will be asked to demonstrate:

   - what you have implemented for each of the levels,
   - If you have implemented level 5 you will also be asked to demonstrate adding your JavaBean(s) to the NetBeans palette and use it/them in another very simple application to show that they can be reused.

   The demonstration will take approximately 15 minutes. As well as demonstrating your application running you will be asked about your code and expected to show a good understanding of it.

   If you have worked in a team you will be assumed to have used pair programming and each member will be expected to demonstrate a good understanding of the code and rationale behind its design. Either member of the pair could be asked to respond to questions related to the submission.

   You will informed by your tutor the date of your acceptance test (demonstration). **You will fail the coursework if you do not attend the demonstration unless you have submitted valid extenuating circumstances.**

3. A report consisting of the following sections. Please use the final report template that is available on Moodle under the Coursework Details and Submission block. **For sections 1 to 6, both members of the team must submit exactly the same – this part is not individual. Section 7 is individual work. Section 8 will be your logbook exercises.**

   Section 1. A brief statement of which of the levels (1 to 6) you have completed (see above).

   Section 2. A UML class diagram of your application.

   Section 3. A concise list of bugs and/or weaknesses in your work (if you don't think there are any then say so). Bugs that are declared in this list will lose you fewer marks than ones that you don't declare!

   Section 4. Documentation of each of the levels you completed as follows.

   Level 1
   - Brief description (up to 200 words) of the ways in which your program is more reliable, flexible and easy to use than the example code you were supplied.

   Level 2
   - A list of the object-oriented features implemented. For each feature note the classes and/or interfaces involved and their relationship to each other.
   - Brief reflection (up to 300 words) on how the object-oriented features implemented improve the design of the system in order to make it easier to maintain and enhance in the future.

   Level 3
   - A list of the classes that you tested using JUnit and the number of tests for each class.

   Level 4

   - For each of the design patterns you implemented:
     - the name of the pattern,
     - a UML class diagram showing the classes and interfaces involved in your implementation of the pattern and the relationships between them,

- a brief reflection (up to 150 words) about how your design pattern has improved the design of the system in order to make it easier to maintain and enhance in the future.

Level 5

- For each of the JavaBeans you implemented:
  - its class name,
  - a brief (up to 50 words) description of its purpose and role,
  - the number of **properties** the JavaBean has,
  - whether or not the **BeanInfo** file causes an icon to be displayed.

Level 6

– For each stretch feature you implemented:
- brief description of the feature (up to 200 words) including how you made use of threads in implementing the feature,
- a list of strengths and weaknesses of your implementation of the feature.

Section 5. Annotated screen shots demonstrating your application. Make sure that the screen shots make clear what you implemented and achieved for each of the levels.

Section 6. Evidence of your use of Git for version control. Include annotated screen shots showing how you made use of Git for version control for the coursework. Make sure you include shots showing dates so that we can see what use you made of Git throughout the period you worked on the coursework.

Section 7. This section is individual and should be written by one person only. Write a brief reflection (from 350 to 500 words) on pair programming.

If you worked as part of a team and thus used pair programming for part B of the coursework, please base this section on your experience. Include discussion of the advantages and disadvantages you experienced in terms of impact on: productivity, errors, learning, time management and any other points you think significant. Remember that pair programming is two programmers working together using one machine it is not just about collaborating with each other. You should comment as to whether you would agree with or suggest changes to the 6 steps suggested by Pea Tyczynska in her article on 'How to pair program effectively in 6 steps' (found at https://gds.blog.gov.uk/2018/02/06/how-to-pair-program-effectively-in-6-steps/).

If you did not work as part of a team and thus did not use pair programming for part B of the coursework, you should base this section on a) your experience of pair programming in the tutorials and b) your research into the advantages and disadvantages of pair programming. Before writing this section you should read at least one journal or conference paper on pair programming (use Google Scholar to search for "pair programming") and reference this from your report. Discuss the advantages that are claimed for pair programming and relate this to your experience during the tutorials. Explain your reasons for not wishing to use pair programming for the coursework.

Section 8. Include a copy of each of the four logbook template documents.

Grading Criteria

Note that this coursework will not be marked anonymously.

Part A – Logbook

Each of the 4 logbooks exercises will be marked out of 3 as follows.

Mark	Requirements
3	All parts completed correctly with at most one minor flaw. Clear and complete report.
2	Nearly all parts completed with up to three minor flaws or one significant flaw. Clear and complete report.
1	Good attempt, but with several major flaws or numerous minor flaws.
0	Little or no attempt.

Part B – Software Development

Marks will be allocated according to the table below. Note that marks shown are out of 100. They will be scaled to be out of 88 and then added to the Logbook mark to give you an overall mark.

To be eligible for the mark in the left-hand column you should at least achieve what is listed in the right-hand column. Note that you may be awarded a lower mark if you don't achieve all the criteria listed. For example, if you achieve all the criteria listed for a 2:1 mark, but have a poor report then your mark might be in the 2:2 range or lower. You must attend a demonstration in order to be eligible to pass this course.

1st > 80%	• Completed levels 1 to 6 implemented to an outstanding standard or above (required features implemented, no obvious bugs, outstanding quality code, highly flexible and maintainable application). • Evidence of regular and frequent use of Git. • Able to show outstanding and thorough knowledge of levels 1 to 6 at the acceptance test demonstration, including discussions of possible alternative implementation choices. • Overall report – outstanding (required sections completed accurately and clearly, easy to read and understand, realistic and insightful reflections). • Section 7 of report (reflection on pair programming) – outstanding (excellent points made about the impact on all of productivity, errors, learning, time management).
1st 70-79%	• Completed levels 1, 2, 3, 4 and 5 to an excellent standard (required features implemented, no obvious bugs, excellent quality code, maintainability enhanced). • Evidence of regular and frequent use of Git. • Able to show excellent knowledge of levels 1 to 5 at the acceptance test demonstration, including justification of design choices. • Overall report – excellent (required sections completed accurately and clearly, easy to read and understand, interesting reflections). • Section 7 of report (reflection on pair programming) – excellent (interesting points made about the impact on all of productivity, errors, learning, time management).

2:1 **60-69%**	• Completed levels 1, 2 and 3 to a very good standard (required features implemented, few if any bugs, good quality code). Good attempt at level 4. • Evidence of regular use of Git. • Able to show very good knowledge of levels 1 to 4 at the acceptance test demonstration. • Overall report – very good (required sections completed accurately and clearly). • Section 7 of report (reflection on pair programming) – very good (good points made about the impact on at least three out of productivity, errors, learning, time management).
2:2 **50-59%**	• Completed levels 1, 2 and 3 to a good standard (required features implemented, few if any bugs). • Evidence of regular use of Git. • Able to show good knowledge of level 1, 2 and 3 at the acceptance test demonstration. • Overall report – good (required sections completed accurately and clearly). • Section 7 of report (reflection on pair programming) – good (addresses impact on at least three out of productivity, errors, learning, time management).
Strong 3rd **45-49%**	• Completed level 1 to a good standard (required features implemented, few if any bugs), reasonable attempt at level two. • Evidence of some attempt to use Git. • Able to show reasonable knowledge of level 1 and level 2 at the acceptance test demo. • Overall report – acceptable (required sections completed). • Section 7 of report (reflection on pair programming) – acceptable (addresses impact on at least two out of productivity, errors, learning, time management).
Weak 3rd **40-44%**	• Completed level 1 to a reasonable standard (required features implemented with a few minor bugs). • Able to show fair knowledge of level 1 at the acceptance test demonstration. • Overall report – acceptable (required sections completed). • Section 7 of report (reflection on pair programming) – acceptable (addresses impact on at least one of productivity, errors, learning, time management).
Fail **35-39%**	• Reasonable attempt at level 1 although maybe buggy. • Able to show adequate knowledge of level 1 at the acceptance test demonstration. • Overall report – mostly completed to a reasonable standard.
Fail **<35%**	• Little or no attempt at level 1 or very buggy. • Not able to show adequate knowledge of level 1 at the acceptance demonstration. • Overall report – mostly in-completed, at an un-acceptable standard or missing.

Assessment Criteria

These are the key points that will be taken into account when marking your work.

Part A – Logbook

For the logbook exercise, marks will be awarded for:

- clarity of report,
- completeness of solutions.

Part B – Software Development

General points that apply to all levels

Reliability of your code. Does it run correctly or does it crash or give incorrect results? Are exceptions handled properly? Faults that you admit on your bug list (see deliverables) will be looked on more kindly than those that are not declared.

Quality of the code. For example: inclusion of meaningful comments, use of sensible naming standards (e.g. for variables and methods) and code layout (e.g. indentation to make the structure of if statements clear)? Follow the Java naming conventions at:
http://www.oracle.com/technetwork/java/javase/documentation/codeconventions-135099.html#367
Lot of duplicate or near duplicate code usually indicates poor design which would benefit from refactoring. Features of the Java language (e.g. accessibility modifiers) and the API should be used appropriately. How easy would it be to add or change features? For example, if making a relatively small enhancement to the program's functionality would require a lot of change to the code, then this is usually an indication of poor design.

Quality of the report. Are all the required sections included and completed properly? Is the report clear, accurate and easy to read? Do the sections that ask you to provide a "reflection" on something show that you have really thought about the issues and have given a balanced view of positive and negative points?

Ability to answer questions at the acceptance test. Both members of the team should be able to:

- Identify how visible behaviour of the program is implemented in the code (e.g. if asked "Where is this validation carried out" they will be able to quickly and accurately find the code)
- Talk through specified fragments of code (e.g. if asked "What does this method do" they will be able to give an overview of its purpose and then go through it line by line explaining its logic and behaviour).
- Explain the use of specified elements of the Java API of language in the code (e.g. if asked "Does the code written for the coursework make use of any interfaces?" be able to answer confidently and point out the use of any interfaces in the code).
- Explain the reasons for specified design decisions and discuss alternative possibilities.
- Note that as team members are expected to have developed the code using pair programming both members of the team should be able to answer questions about any part of the code. An answer such as

"My partner wrote that bit of code so I can't answer questions about it." may negatively affect your individual mark.

Points specific to particular levels

Level 2 OO Design:

Scalability and reusability of class designs. The design approach taken. Is the software easy to maintain and extend? Are the classes reusable? **UML diagram:** Does the diagram use correct UML notation? Is it detailed enough to show the important aspects of the design but not so detailed that the important information is lost in a mass of detail. Is the diagram an accurate representation of your implementation?

Level 3 JUnit testing:

Do the tests cover a good range of conditions including exceptions? Even a fairly simple method normally needs quite a few tests cases to check it properly.

Level 4 Design Patterns:

Are they implemented correctly? Are they useful in the way they have been used (i.e. to they improve the current design or add to scalability of the code?

Level 5 Component implementation:

Adherence to standards and usefulness and reusability of the component(s) that you've created. Would another programmer want, and be able, to reuse your component(s) in other applications? Factors that contribute to reusability include: flexibility (e.g. plenty of get and set methods to allow its properties to be configured), good documentation, having a clear and specific purpose, useful features such as validation.

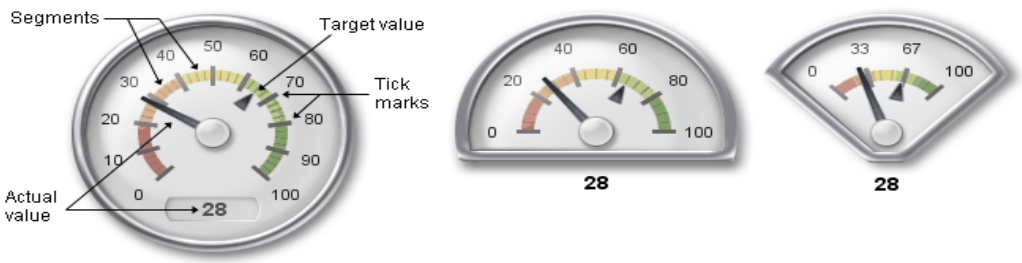
Appendix A

This section provides ideas for different dials and vertical/horizontal bars. Remember this coursework is not about graphics – you can use different variations of the simple dials and bars provided in the DashboardDemo project.

General dashboard indicators – examples of bars, switches, and dials.

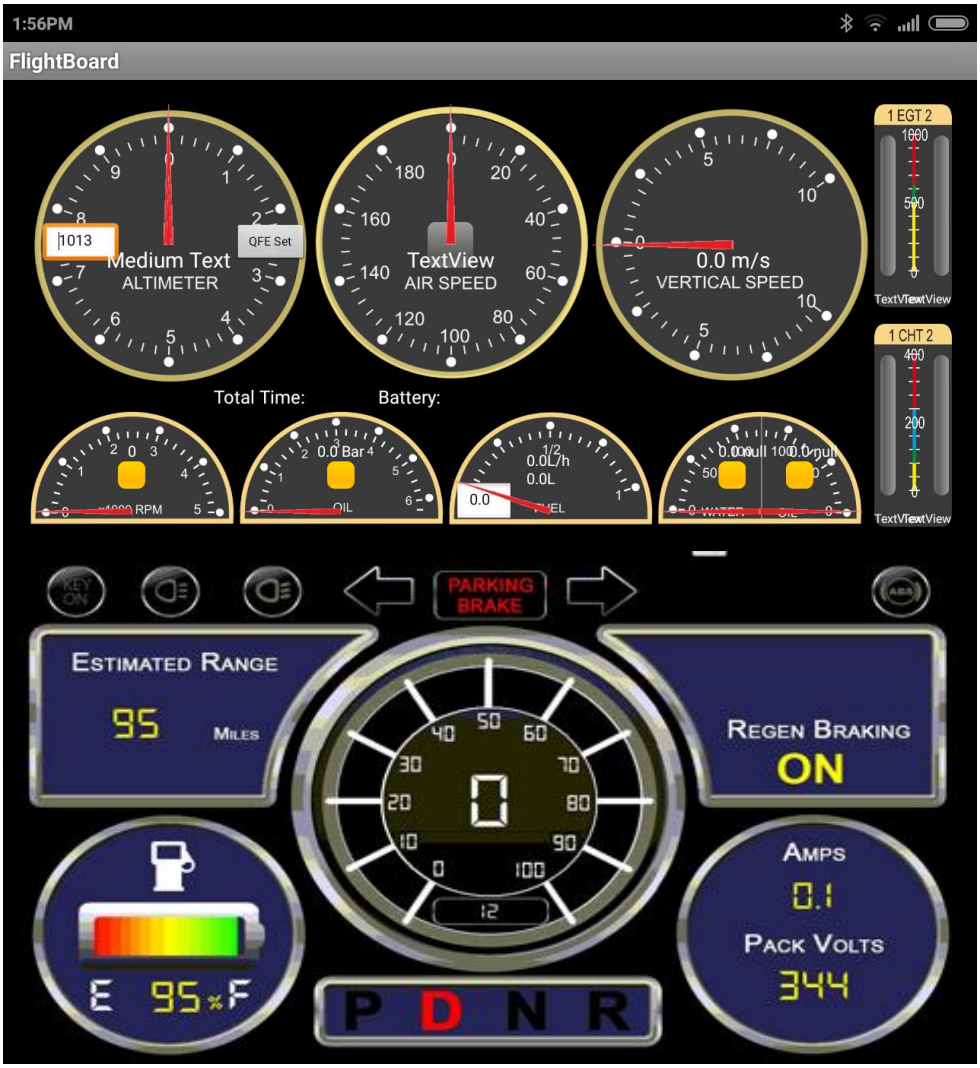
Dial types

Type=Full Type=Half Type=Quarter



Train simulator dashboards ideas

Aircraft dashboard ideas



Vertical bars – server performance