Please refer to the online lesson plan on the DT Hub to access all website links and additional resources.

## Learning hook

Illustrate the basic concepts of OO programming by having students play Tetris – a game with which they are likely to be familiar.

1. Have students play classic Tetris online.
2. After playing the game Tetris have students participate in a class discussion to determine:
   * which different categories of objects might be created in designing this game?  
     (playing board, playing pieces, high-score gallery)  
     Explain that we call these *classes* in OOP.
   * what would be the *attributes* or properties of each of these classes?  
     (playing board: different colours, dimensions, music choice etc)  
     (playing pieces: 7 different shapes, colour, rotation)
   * what methods or *behaviours*might be invented for these classes?  
     (playing board: gravity)  
     (playing pieces: falling, moving sideways and rotating)
   * One of our classes is the family of playing pieces. What different ones might be created?  
     (one playing board as an instance of the Playing board class; 7 differently shaped blocks as instances of the Playing pieces class, called Tetrominos, which are the required playing objects)  
     In OOP these are called different *instances*of the ‘playing piece’ class.

**Car company analogy**

The analogy of a car company producing different models in their factory may assist in explaining OOP principles and advantages.  
  
Each model of car designed by a car company can be considered a separate class described by the factory’s blueprints for each class, designed before any of the cars exist.  
   
One of these classes might be a hatchback design. This class will have certain properties, such as colour, hatchback door, air conditioning (yes or no), luxury additions (yes or no) and so on. It will also have certain behaviours that it owns such as a hatchback door that can be lifted up.  
   
When the first hatchback is made, it is said to be an instance of the hatchback class. All new hatchbacks will share the same properties and behaviours designed for the hatchback class.  
  
There will be other classes too, such as a sports model, with its own properties and behaviours.  
  
In fact, it might be a good idea to create a kind of ‘super class’ called ‘car’ to which all these other classes belong and assign to it all the properties and behaviours that all models possess, to save us repeating them every time a new one comes out. After all, all our models will have headlights, steering wheels, seats and engines!  
  
When we design a complex program, it is much easier if similar objects are treated as classes, with properties and behaviours. Once we have done this, we don’t need to keep thinking about the details, as we have made a component we can use when we need to, knowing it will always behave the same way.  
  
Functions perform a similar role in programming as we write them once and just call them when we need that behaviour, but OOP takes that a step further. Instead of having lots of different functions scattered through our programs, making it easy for a programmer to get lost, OOP collects all functions belonging to the same object and groups them as a class.

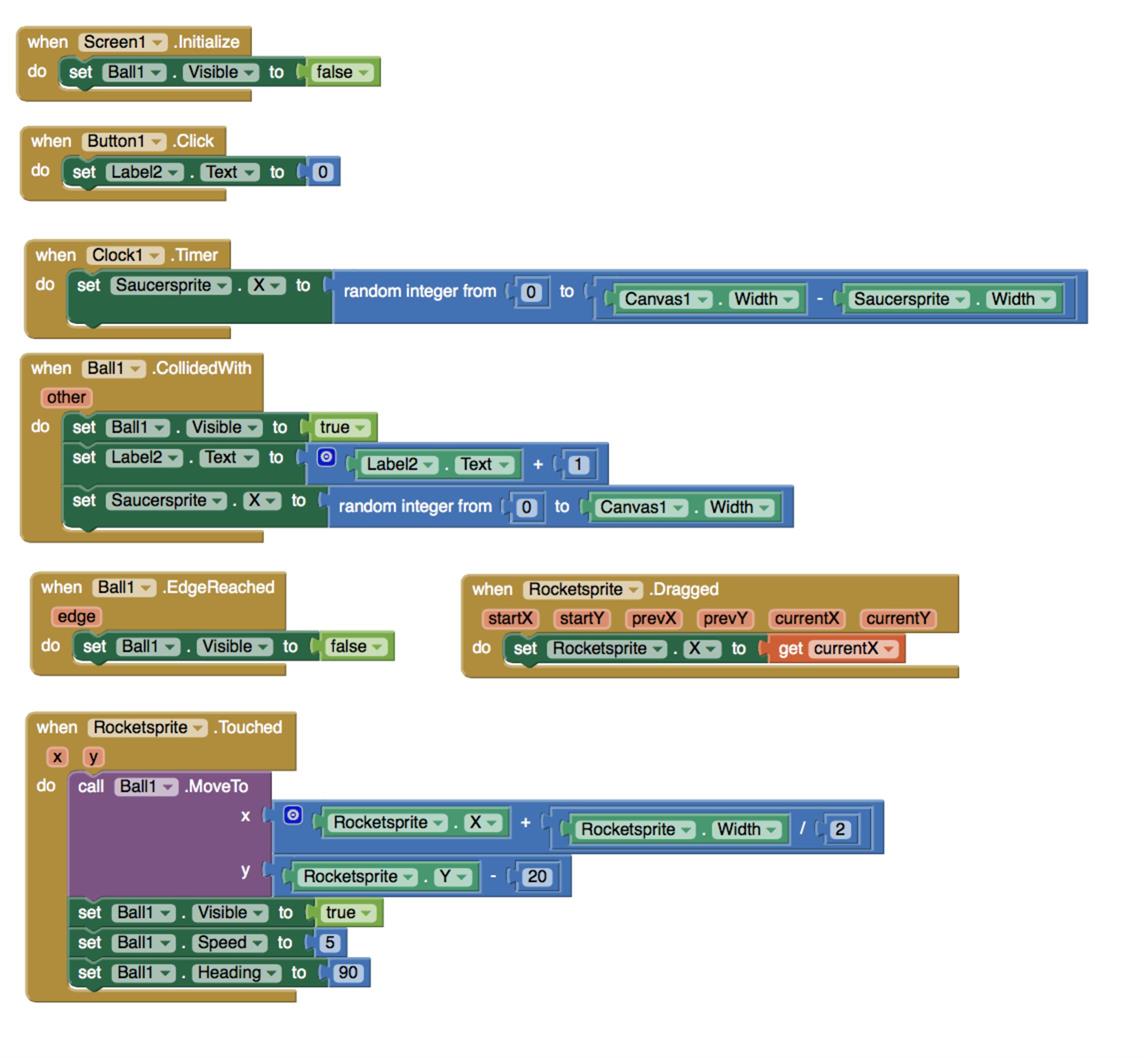
## Learning map and outcomes

Students learn about Object Oriented Programming (OOP) methods using classes as a useful approach in the development of more complex programs, where separate objects are defined as having attributes and behaviours.

## Learning input

**Game-based visual programming approach**

This approach is suitable for all students. In this approach, the teacher introduces OOP principles using one of the visual programming tools listed in Resources.  
  
(See *Using a text-based language* for an alternative text-based programming approach suitable for more able students)

1. Space Invaders tutorial   
   This tutorial uses MIT App Inventor, which runs on Mac or Windows. Working programs can be viewed using the inbuilt Android emulator. MIT App Inventor is OOP based and it is free.  
   Students will build a Space Invaders game using the clearly explained intermediate tutorial provided by MIT ( see online lesson plan on the DT Hub to access all website links).  
   The major emphasis will be on visual programming where the teacher emphasises the use and advantages of classes, attributes and methods/behaviours.  
   MIT App Inventor uses these same principles but calls its classes **Components**(OOP: *Classes*),which have **Properties**(OOP: *Attributes*) and which perform functions built with jigsaw pieces that are called **Blocks**(OOP: *Behaviours*).  
   Although this tutorial produces a very simple game, once the concepts are understood, students can progress to more advanced projects.
2. Ask the students to follow these steps:  
   1. Save the two .png images of the rocket and the flying saucer to the desktop.
   2. Set up the Components for the game’s interface.
   3. Saucer speeds up its movement
   4. Add rocket behaviours
   5. Add bullet behaviours
   6. Add button method
   7. Add flying saucer behaviours
   8. Test the game on the emulator on Mac or Windows or on an Android phone
3. When working through the tutorial draw students’ attention to these OOP aspects:  
   1. **RocketSprite**, **SaucerSprite**and **BallSprite**are all instances of the one Sprite superclass: but RocketSprite and SaucerSprite are instances of a subclass called **ImageSprites**which get their appearance from imported image files: BallSprite is another subclass whose appearance can only be changed by varying its Paintcolour and Radius properties.   
      (Note: This is an example of *polymorphism*in OOP, defined as the ability to redefine methods for derived classes.)
   2. Point out the

Screen1.initialize

method as an example of a behaviour which takes place at initialisation of the screen class.

* 1. Point out that the right-hand Properties column lists all properties for any given component.

1. Introduce students to the following website: *Components – App Inventor for Android*   
   This is where all App Inventors components are described in detail along with their properties, their methods (behaviours) and the events to which they respond.

## Learning construction

1. Review the OOP features that have been used in the tutorial.   
   Students can then choose to add features to this game, tackle another tutorial or design their own app in App Inventor, starting from scratch.
2. Suggest students add additional features to their Space Invaders game. They could:
   * improve the smoothness of movement
   * have flying saucers (with 3 lives) start from the top and gradually move downwards, with collision with rocket ending the game
   * add different objects in addition to the flying saucer
   * add music
   * add backgrounds.
3. Students could also investigate other suitable tutorials listing on the Tutorials for App Inventor website (MIT).
4. Students could choose to develop their own game.

## Learning demo

Quiz students to elicit OOP characteristics as they present their game via a short movie. (Short screen capture movies are easier than loading each game in turn to an App Inventor’s emulator.)  
  
Students show their creations to the class and detail the OOP features: classes, attributes and behaviours.

## Curriculum links

| Links with Digital Technologies Curriculum Area | |
| --- | --- |
| **Strand** | **Content Description** |
| **Processes and Production Skills** | Implement, modify and debug modular programs, applying selected algorithms and data structures, including in an object-oriented programming language (AC9TDI10P09) . |

## Assessment

|  | **Quantity of Knowledge** | | | **Quality of Understanding** |
| --- | --- | --- | --- | --- |
| **Game creation** | No evidence of understanding | Student is able to develop the correct components/classes for the design | Student is able to build the screen design using components/classes | Student is able to design methods for the components/classes | Student game is tested and fulfils all requirements and successfully demonstrates at least one additional feature |
| **Optional Score** | 0 | 1 | 2 | 3 | 4 |