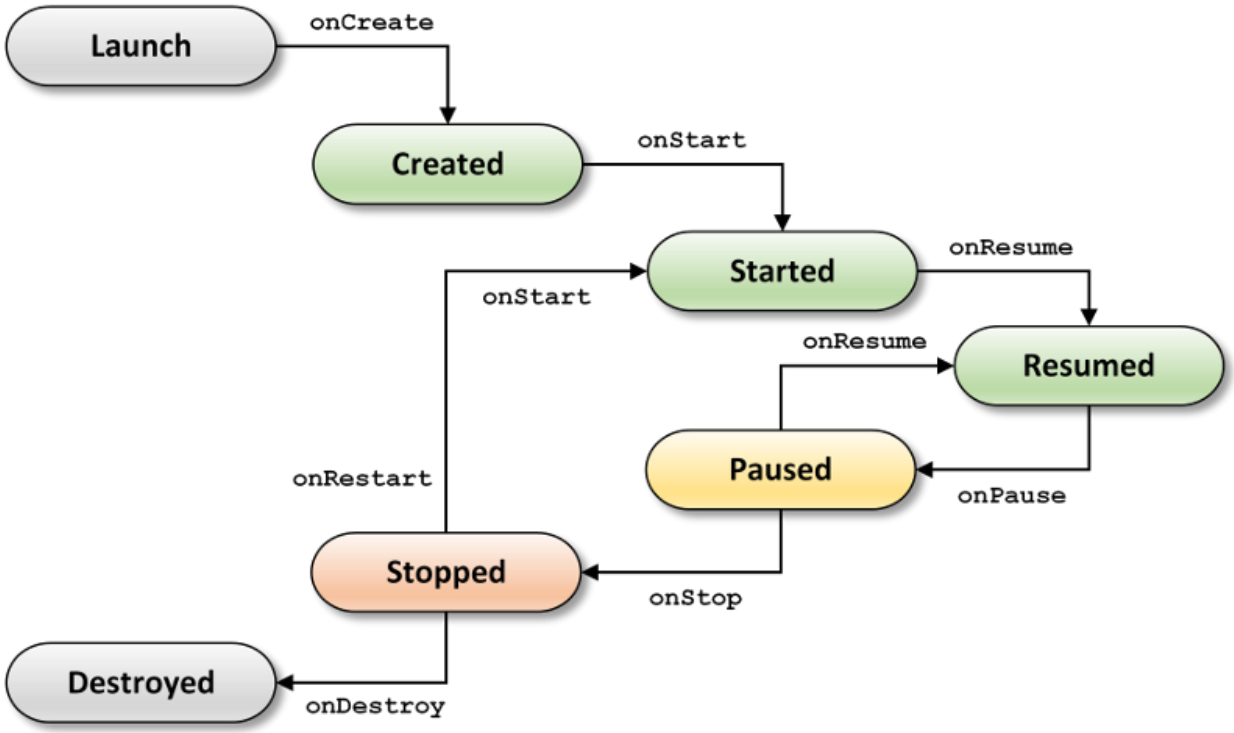
Contribution from Eli Tatro, Team Venture

Here is the problem and here is how we solved it.
There are two things that need to be looked at when you are coding your APP.
First we need to look at the life cycle of an APP.
Second we need to solve the Screen Tilt problem
And third we need to be able to save our APP, exit completely, and reload it.

First is the life cycle of an APP.



When you tilt the screen on your APP to make it horizontal, the app goes through this faze to onPause, if you do not save your object (mainly your facade) it will restart the activity you are on and refresh all your variables, as if the button to get to that activity had been pressed.

So in order to make sure that this does not happen, and your user can tilt their screen at any time, you have two options, first: Lock them out of the ability to rotate screen, second save the state of your object to the bundle with something like this:

```
@Override
  public void onSaveInstanceState(Bundle savedInstanceState){
    savedInstanceState.putSerializable("model", model);
    super.onSaveInstanceState(savedInstanceState);

  }
  @Override
  public void onRestoreInstanceState(Bundle savedInstanceState){
    super.onRestoreInstanceState(savedInstanceState);
    model = (AdventureGameModelFacade) savedInstanceState.getSerializable("model");
  }
```

At the beginning of your activities code, just check if the bundle is null or not. If it is, create your façade normally, if not, load it from the bundle.
The bundle will save the state of the activity and overwriting onResume will tell the bundle to save your instance state.

Now on to saving. The confusion exists with the Bundle Object, after many hours of testing we came to the conclusion that the Bundle is only for switching between activities and states. When the user terminates the program, and onDestroy is called, the bundle is part of what gets destroyed.

So first step is, SERIALIZE all portions of your code that have objects that maintain a state, for us this included all objects that our Fascade utilized, cavesite, player, item. And ALL OF THEIR subclasses, must implement Serializable.

Lets not forget that when we serialize an object, that it tears everything in the object down to its serialized code. (Note: in the real world, this is not a good practice, code is ever evolving and once something is serialized the format of the object and its reliant classes must be EXACTLY the same.) Java has built in methods for doing reliant serialization but that rabbit hole is beyond our scope. So, make sure everything implements serializable.

The next thing we need to do is create a “file” that will store our serialized main object, to be called upon later when load is pressed. For this we use a specific type of stream, you are going to need to import some libraries, but that is ok. The stream is an object stream that takes a SERIALIZABLE instance of an object, creates a save file on the phone that is set to be private only for the application. (So when you uninstall the APP it takes the save file with it.) What better thing to serialize than the instance of Factory? (If you have not fallowed the format of code outlined in class, at this point you are sunk.)

For Save and Load

```
public void Save(){
        Log.d("Save", "saving object");
    try{
      File gameInfo = new File("save.bin");
      FileOutputStream fileOutputStream = openFileOutput("save.bin", Context.MODE_PRIVATE);
      ObjectOutputStream objectOutputStream = new ObjectOutputStream(fileOutputStream);
      objectOutputStream.writeObject(model);
      objectOutputStream.close();
      Log.d("Save", "object saved");
    }catch(FileNotFoundException e){
      Log.e("Save", e.getMessage());
    }catch(IOException e){
      Log.e("Save", e.getMessage());
    }
  }

  public Object Load(){
    Log.d("Load", "loading object");
    Object object = null;
    try{
      FileInputStream fileInputStream = openFileInput("save.bin");
      ObjectInputStream objectInputStream = new ObjectInputStream(fileInputStream);
      object = objectInputStream.readObject();
      objectInputStream.close();
      Log.d("Load", "object loaded");
    }catch(FileNotFoundException e){
      Log.e("Load", e.getMessage());
    }catch(StreamCorruptedException e){
      Log.e("Load", e.getMessage());
    }catch(Exception e){
      Log.e("Load", e.getMessage());
    }
    return object;
  }
```

Now you have your methods, but when do you call them?
You will notice that Load returns an object of type OBJECT.
You can call onCreate, with your load button as well as with your Level1 selection! Use the same logic you used for selecting difficulty 1.
But pass in a flag for loading. The button and passing logic looks like this:

```
public void clickLoad(View view){
    Intent intent = new Intent(this, LevelOneScreen.class);
    intent.putExtra("key", 1);
    startActivity(intent);
    finish();
}
```

This passes an integer bundle into the Activity of loading the levelOneScreen.
And the logic for on create of your Level 1 activity needs to have a check to see if it is loading from Load or from start. We have passed in an Extra called “key” this will be what we will check to see what button was used to call on create. Create your model fascade as usual at the top of on create, but add similar logic such as this:

Where model is the instance of your Façade.

```
Bundle b = getIntent().getExtras();
        if (b != null) {
            int loadFlag = b.getInt("key");
            if (loadFlag == 1) {
                model = (AdventureGameModelFacade) Load();
                loadFlag = 0; ← Do not forget to reset your flag, or else every time you call on create, it will load the saved game!
            }
        }
```

Remember that Load returns an Object type, and Object is the granddaddy of all objects in java, so cast it and enter a contract that Load will be guaranteeing a AdventureGameModel Façade, and you are set! Note in order for this to work, all your instance states, (what room items were left in, what player is holding. Need to all be handled in the Façade, if you fallowed the instructions for A4 up to this point, you are all good.)