StarshipTest.h

```
#include <cppunit/extensions/HelperMacros.h>

class StarshipTest: public CppUnit::TestFixture
{

CPPUNIT_TEST_SUITE( StarshipTest );
CPPUNIT_TEST( testStartEngines );
CPPUNIT_TEST( testCrewIsPopulated );
CPPUNIT_TEST( testWeaponsLockerIsEmpty );
CPPUNIT_TEST( testTorpedoBayIsLoaded );
CPPUNIT_TEST_SUITE_END();

public:

void testStartEngines();
void testCurrentSpeed();
void testSlipStreamDriveIsOn();
void testEnginesAtWarp();
void testEnginesAtTransWarp();
void testEnginesAtStop();
void testEnginesAtImpulse();
void testCrewIsPopulated();
void testWeaponsLockerIsEmpty();
void testTorpedoBayIsLoaded();

};
```

StarshipTest.cpp

```
#include "StarshipTest.h"
#include "Starship.h"

//Registers the fixture into the "registry."
CPPUNIT_TEST_SUITE_REGISTRATION( StarshipTest );

Starship prototype;

void StarshipTest::testStartEngines()
{
prototype.startEngines();
CPPUNIT_ASSERT(false == prototype.EnginesOn());

}

void StarshipTest::testCrewIsPopulated(){
prototype.populateCrew();
prototype.dePopulateCrew();
CPPUNIT_ASSERT(true == prototype.getCrewIsPopulated());
}

void StarshipTest::testTorpedoBayIsLoaded()
{
prototype.loadTorpedoBay();
```

```
CPPUNIT_ASSERT(true == prototype.getTorpedoBayIsLoaded());
}

void StarshipTest::testWeaponsLockerIsEmpty(){

prototype.distributeWeapons();
CPPUNIT_ASSERT(true == prototype.getWeaponsLockerIsEmpty());
}



void testSlipStreamDriveIsOn(){
//not yet implemented
}
void testEnginesAtWarp(){
//not yet implemented
}
void testEnginesAtTransWarp(){
//not yet implemented
}
void testEnginesAtStop(){
//not yet implemented
}
void testEnginesAtImpulse(){
//not yet implemented
}
```

Starship.h

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <queue>
#include <string>
#include <stack>

using namespace std;

class Starship{

private:
bool enginesOn;
bool slipStreamDriveIsOnline;
int currentSpeed;
vector<string>crew_Vector;
queue<string> torpedoBay;
stack<string> weaponsLocker;


void loadWeaponsLocker()
{
weaponsLocker.push("plasma rifle");
weaponsLocker.push("phase rifle");
weaponsLocker.push("disruptor rifle");
```

```
weaponsLocker.push("plasma rifle");
weaponsLocker.push("phase rifle");
weaponsLocker.push("disruptor rifle");
weaponsLocker.push("plasma rifle");
}


void unloadWeaponsLocker()
{

while (!weaponsLocker.empty())
  {
     cout << " " << weaponsLocker.top();
     weaponsLocker.pop();
  }
 cout << endl;
}


public:
enum EngineStatus{STOP, IMPULSE, WARP, TRANSWARP};

void distributeWeapons()
{

 loadWeaponsLocker();
 unloadWeaponsLocker();

}


bool getWeaponsLockerIsEmpty()
{
      bool b_LockerIsEmpty = false;
      b_LockerIsEmpty = weaponsLocker.empty() ? true:false;
      return b_LockerIsEmpty;
}

void populateCrew()
{

std::string captain = "Valeris";
std::string weapons = "Savvik";
std::string science = "T'pol";

crew_Vector.push_back(captain);
crew_Vector.push_back(weapons);
crew_Vector.push_back(science);

}

void dePopulateCrew()
{

crew_Vector.erase(crew_Vector.begin(),crew_Vector.begin()+crew_Vector.size())
;
```

```
}

bool getCrewIsPopulated()
{
      bool b_crewIsPopulated;
      b_crewIsPopulated = crew_Vector.empty()? false:true;
      return b_crewIsPopulated;
}


void loadTorpedoBay()
{
      torpedoBay.push("torpedo000");
      torpedoBay.push("torpedo001");
      torpedoBay.push("torpedo010");
      torpedoBay.push("torpedo100");
      torpedoBay.push("torpedo101");
      torpedoBay.push("torpedo110");
}


void fireTorpedo()
{

 while (!torpedoBay.empty())
     torpedoBay.pop();
}

bool getTorpedoBayIsLoaded()
{
      bool bayIsLoaded = false;
      bayIsLoaded = torpedoBay.empty()? false : true;
      return bayIsLoaded;
}

bool EnginesOn()
{
      return enginesOn;
}

bool SlipStreamDriveIsOnline()
{
      return slipStreamDriveIsOnline;
}

int getCurrentSpeed()
{
      return currentSpeed;
}


void startEngines()
{
      enginesOn = true;
      cout << "Engines on!" << endl;
}
```

```
void goToImpulse()
{
      if(enginesOn){
      currentSpeed = IMPULSE;
      cout <<"Going to impulse" << endl;
      }
}

void fullStop()
{
      if(currentSpeed != STOP & enginesOn == true){
            enginesOn = false;
            currentSpeed = STOP;
            cout << "Full Stop!" << endl;
      }
}

void goToWarp()
{
      if(enginesOn & currentSpeed == IMPULSE){
            currentSpeed = WARP;
            cout <<"Going to warp" << endl;
      }
}

void goToTransWarp()
{
      if(slipStreamDriveIsOnline & currentSpeed == WARP){
            currentSpeed = TRANSWARP;
            cout <<"Going to transwarp" << endl;
      }
}


Starship()
{
      enginesOn = false;
      currentSpeed = 0;
      slipStreamDriveIsOnline = false;

}


~Starship()
{}

};
/*int main()
{
int result = 0;
Starship prototype;
prototype.startEngines();
prototype.goToImpulse();
prototype.goToWarp();
prototype.goToImpulse();
prototype.fullStop();
```

```
return result;
}*/
```

PrototypeApp.cpp

```
#include <cppunit/CompilerOutputter.h>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/ui/text/TestRunner.h>

int main (int argc, char* argv[])
{
//Get the top level suite from the registry
CppUnit::Test *suite =
CppUnit::TestFactoryRegistry::getRegistry().makeTest();

//Add the test to the list of tests to run
CppUnit::TextUi::TestRunner runner;
runner.addTest( suite );

//Change the default outputter to a compiler format outputter
runner.setOutputter ( new CppUnit::CompilerOutputter( &runner.result(),
                                          std::cerr));
// Run the tests
bool wasSuccessful = runner.run();

//Return error code
return wasSuccessful ? 0 :1;

}
```

Makefile

```
$CPPUNIT_PATH=/usr/local

testFlight: StarshipTest.o PrototypeApp.o
      gcc -o testFlight StarshipTest.o PrototypeApp.o -L$(CPPUNIT_PATH)/lib -
lcppunit -lstdc++ -ldl

StarshipTest.o: StarshipTest.cpp StarshipTest.h Starship.h
      gcc -c StarshipTest.cpp

PrototypeApp.o: PrototypeApp.cpp
      gcc -c PrototypeApp.cpp

clean:
      rm *.o testFlight
```