```
namespace WarMachines.Engine
{
    using System;
    using System.Collections.Generic;
    using System.Linq;

    using WarMachines.Interfaces;

    public class Command : ICommand
    {
        private const char SplitCommandSymbol = ' ';

        private string name;
        private IList<string> parameters;

        private Command(string input)
        {
            this.TranslateInput(input);
        }

        public string Name
        {
            get
            {
                return this.name;
            }

            private set
            {
                if (string.IsNullOrEmpty(value))
                {
                    throw new ArgumentNullException("Name cannot be null or empty.");
                }

                this.name = value;
            }
        }

        public IList<string> Parameters
        {
            get
            {
                return this.parameters;
            }

            private set
            {
                if (value == null)
                {
                    throw new ArgumentNullException("List of strings cannot be null.");
                }

                this.parameters = value;
            }
        }

        public static Command Parse(string input)
        {
            return new Command(input);
        }

        private void TranslateInput(string input)
        {
            var indexOfFirstSeparator = input.IndexOf(SplitCommandSymbol);

            this.Name = input.Substring(0, indexOfFirstSeparator);
            this.Parameters = input.Substring(indexOfFirstSeparator + 1).Split(new[] { SplitCommandSymbol }, ↙
      StringSplitOptions.RemoveEmptyEntries);
        }
    }
}
namespace WarMachines.Engine
{
    using WarMachines.Interfaces;
    using WarMachines.Machines;

    public class MachineFactory : IMachineFactory
    {
        public IPilot HirePilot(string name)
        {
            return new Pilot(name);
        }

        public ITank ManufactureTank(string name, double attackPoints, double defensePoints)
        {
            return new Tank(name, attackPoints, defensePoints);
        }

        public IFighter ManufactureFighter(string name, double attackPoints, double defensePoints, bool ↙
    stealthMode)
        {
            return new Fighter(name, attackPoints, defensePoints, stealthMode);
        }
    }
}
namespace WarMachines.Engine
{
    using System;
    using System.Collections.Generic;
    using System.Text;

    using WarMachines.Interfaces;

    public sealed class WarMachineEngine : IWarMachineEngine
    {
        private const string InvalidCommand = "Invalid command name: {0}";
        private const string PilotHired = "Pilot {0} hired";
        private const string PilotExists = "Pilot {0} is hired already";
        private const string TankManufactured = "Tank {0} manufactured - attack: {1}; defense: {2}";
        private const string FighterManufactured = "Fighter {0} manufactured - attack: {1}; defense: {2}; ↙
    stealth: {3}";
        private const string MachineExists = "Machine {0} is manufactured already";
        private const string MachineHasPilotAlready = "Machine {0} is already occupied";
        private const string PilotNotFound = "Pilot {0} could not be found";
        private const string MachineNotFound = "Machine {0} could not be found";
        private const string MachineEngaged = "Pilot {0} engaged machine {1}";
        private const string InvalidMachineOperation = "Machine {0} does not support this operation";
        private const string FighterOperationSuccessful = "Fighter {0} toggled stealth mode";
        private const string TankOperationSuccessful = "Tank {0} toggled defense mode";
        private const string InvalidAttackTarget = "Tank {0} cannot attack stealth fighter {1}";
        private const string AttackSuccessful = "Machine {0} was attacked by machine {1} - current health: ↙
    {2}";

        private static readonly WarMachineEngine SingleInstance = new WarMachineEngine();

        private IMachineFactory factory;
        private IDictionary<string, IPilot> pilots;
        private IDictionary<string, IMachine> machines;

        private WarMachineEngine()
        {
            this.factory = new MachineFactory();
            this.pilots = new Dictionary<string, IPilot>();
            this.machines = new Dictionary<string, IMachine>();
        }

        public static WarMachineEngine Instance
        {
            get
            {
                return SingleInstance;
            }
        }

        public void Start()
        {
            var commands = this.ReadCommands();
            var commandResult = this.ProcessCommands(commands);
            this.PrintReports(commandResult);
        }

        private IList<ICommand> ReadCommands()
        {
            var commands = new List<ICommand>();

            var currentLine = Console.ReadLine();

            while (!string.IsNullOrEmpty(currentLine))
            {
                var currentCommand = Command.Parse(currentLine);
                commands.Add(currentCommand);

                currentLine = Console.ReadLine();
            }

            return commands;
        }

        private IList<string> ProcessCommands(IList<ICommand> commands)
        {
            var reports = new List<string>();

            foreach (var command in commands)
            {
                string commandResult;

                switch (command.Name)
                {
                    case "HirePilot":
                        var pilotName = command.Parameters[0];
                        commandResult = this.HirePilot(pilotName);
                        reports.Add(commandResult);
                        break;

                    case "Report":
                        var pilotReporting = command.Parameters[0];
                        commandResult = this.PilotReport(pilotReporting);
                        reports.Add(commandResult);
                        break;

                    case "ManufactureTank":
                        var tankName = command.Parameters[0];
                        var tankAttackPoints = double.Parse(command.Parameters[1]);
                        var tankDefensePoints = double.Parse(command.Parameters[2]);
                        commandResult = this.ManufactureTank(tankName, tankAttackPoints, tankDefensePoints);
                        reports.Add(commandResult);
                        break;

                    case "DefenseMode":
                        var defenseModeTankName = command.Parameters[0];
                        commandResult = this.ToggleTankDefenseMode(defenseModeTankName);
                        reports.Add(commandResult);
                        break;

                    case "ManufactureFighter":
                        var fighterName = command.Parameters[0];
                        var fighterAttackPoints = double.Parse(command.Parameters[1]);
                        var fighterDefensePoints = double.Parse(command.Parameters[2]);
                        var fighterStealthMode = command.Parameters[3] == "StealthON" ? true : false;
                        commandResult = this.ManufactureFighter(fighterName, fighterAttackPoints,          ↙
    fighterDefensePoints, fighterStealthMode);
                        reports.Add(commandResult);
                        break;

                    case "StealthMode":
                        var stealthModeFighterName = command.Parameters[0];
                        commandResult = this.ToggleFighterStealthMode(stealthModeFighterName);
                        reports.Add(commandResult);
                        break;

                    case "Engage":
```

```
                    var selectedPilotName = command.Parameters[0];
                    var selectedMachineName = command.Parameters[1];
                    commandResult = this.EngageMachine(selectedPilotName, selectedMachineName);
                    reports.Add(commandResult);
                    break;

                case "Attack":
                    var attackingMachine = command.Parameters[0];
                    var defendingMachine = command.Parameters[1];
                    commandResult = this.AttackMachines(attackingMachine, defendingMachine);
                    reports.Add(commandResult);
                    break;

                default:
                    reports.Add(string.Format(InvalidCommand, command.Name));
                    break;
            }
        }

        return reports;
    }

    private void PrintReports(IList<string> reports)
    {
        var output = new StringBuilder();

        foreach (var report in reports)
        {
            output.AppendLine(report);
        }

        Console.Write(output.ToString());
    }

    private string HirePilot(string name)
    {
        if (this.pilots.ContainsKey(name))
        {
            return string.Format(PilotExists, name);
        }

        var pilot = this.factory.HirePilot(name);
        this.pilots.Add(name, pilot);

        return string.Format(PilotHired, name);
    }

    private string ManufactureTank(string name, double attackPoints, double defensePoints)
    {
        if (this.machines.ContainsKey(name))
        {
            return string.Format(MachineExists, name);
        }

        var tank = this.factory.ManufactureTank(name, attackPoints, defensePoints);
        this.machines.Add(name, tank);

        return string.Format(TankManufactured, name, attackPoints, defensePoints);
    }

    private string ManufactureFighter(string name, double attackPoints, double defensePoints, bool       ↙
stealthMode)
    {
        if (this.machines.ContainsKey(name))
        {
            return string.Format(MachineExists, name);
        }

        var fighter = this.factory.ManufactureFighter(name, attackPoints, defensePoints, stealthMode);
        this.machines.Add(name, fighter);

        return string.Format(FighterManufactured, name, attackPoints, defensePoints, stealthMode == true ↙
 ? "ON" : "OFF");
    }
```

```
        private string EngageMachine(string selectedPilotName, string selectedMachineName)
        {
            if (!this.pilots.ContainsKey(selectedPilotName))
            {
                return string.Format(PilotNotFound, selectedPilotName);
            }

            if (!this.machines.ContainsKey(selectedMachineName))
            {
                return string.Format(MachineNotFound, selectedMachineName);
            }

            if (this.machines[selectedMachineName].Pilot != null)
            {
                return string.Format(MachineHasPilotAlready, selectedMachineName);
            }

            var pilot = this.pilots[selectedPilotName];
            var machine = this.machines[selectedMachineName];

            pilot.AddMachine(machine);
            machine.Pilot = pilot;

            return string.Format(MachineEngaged, selectedPilotName, selectedMachineName);
        }

        private string AttackMachines(string attackingMachineName, string defendingMachineName)
        {
            if (!this.machines.ContainsKey(attackingMachineName))
            {
                return string.Format(MachineNotFound, attackingMachineName);
            }

            if (!this.machines.ContainsKey(defendingMachineName))
            {
                return string.Format(MachineNotFound, defendingMachineName);
            }

            var attackingMachine = this.machines[attackingMachineName];
            var defendingMachine = this.machines[defendingMachineName];

            if (attackingMachine is ITank && defendingMachine is IFighter && (defendingMachine as IFighter). ↙
    StealthMode)
            {
                return string.Format(InvalidAttackTarget, attackingMachineName, defendingMachineName);
            }

            attackingMachine.Targets.Add(defendingMachineName);

            var attackPoints = attackingMachine.AttackPoints;
            var defensePoints = defendingMachine.DefensePoints;

            var damage = attackPoints - defensePoints;

            if (damage > 0)
            {
                var newHeathPoints = defendingMachine.HealthPoints - damage;

                if (newHeathPoints < 0)
                {
                    newHeathPoints = 0;
                }

                defendingMachine.HealthPoints = newHeathPoints;
            }

            return string.Format(AttackSuccessful, defendingMachineName, attackingMachineName,              ↙
    defendingMachine.HealthPoints);
        }

        private string PilotReport(string pilotReporting)
        {
            if (!this.pilots.ContainsKey(pilotReporting))
            {
                return string.Format(PilotNotFound, pilotReporting);
            }

            return this.pilots[pilotReporting].Report();
        }

        private string ToggleFighterStealthMode(string stealthModeFighterName)
        {
            if (!this.machines.ContainsKey(stealthModeFighterName))
            {
                return string.Format(MachineNotFound, stealthModeFighterName);
            }

            if (this.machines[stealthModeFighterName] is ITank)
            {
                return string.Format(InvalidMachineOperation, stealthModeFighterName);
            }

            var machineAsFighter = this.machines[stealthModeFighterName] as IFighter;
            machineAsFighter.ToggleStealthMode();

            return string.Format(FighterOperationSuccessful, stealthModeFighterName);
        }

        private string ToggleTankDefenseMode(string defenseModeTankName)
        {
            if (!this.machines.ContainsKey(defenseModeTankName))
            {
                return string.Format(MachineNotFound, defenseModeTankName);
            }

            if (this.machines[defenseModeTankName] is IFighter)
            {
                return string.Format(InvalidMachineOperation, defenseModeTankName);
            }

            var machineAsFighter = this.machines[defenseModeTankName] as ITank;
            machineAsFighter.ToggleDefenseMode();

            return string.Format(TankOperationSuccessful, defenseModeTankName);
        }
    }
}
namespace WarMachines.Interfaces
{
    using System.Collections.Generic;

    public interface ICommand
    {
        string Name { get; }

        IList<string> Parameters { get; }
    }
}
namespace WarMachines.Interfaces
{
    public interface IFighter : IMachine
    {
        bool StealthMode { get; }

        void ToggleStealthMode();
    }
}
namespace WarMachines.Interfaces
{
    using System.Collections.Generic;

    public interface IMachine
    {
        string Name { get; set; }

        IPilot Pilot { get; set; }
        double HealthPoints { get; set; }

        double AttackPoints { get; }

        double DefensePoints { get; }

        IList<string> Targets { get; }

        void Attack(string target);

        string ToString();
    }
}
namespace WarMachines.Interfaces
{
    public interface IMachineFactory
    {
        IPilot HirePilot(string name);

        ITank ManufactureTank(string name, double attackPoints, double defensePoints);

        IFighter ManufactureFighter(string name, double attackPoints, double defensePoints, bool          ↙
    stealthMode);
    }
}
namespace WarMachines.Interfaces
{
    public interface IPilot
    {
        string Name { get; }

        void AddMachine(IMachine machine);

        string Report();
    }
}
namespace WarMachines.Interfaces
{
    public interface ITank : IMachine
    {
        bool DefenseMode { get; }

        void ToggleDefenseMode();
    }
}
namespace WarMachines.Interfaces
{
    using System.Collections.Generic;

    public interface IWarMachineEngine
    {
        void Start();
    }
}
namespace WarMachines.Machines
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using WarMachines.Interfaces;

    public class Fighter : Machine, IFighter
    {
        #region Fields
        private const double InitialHelathPoints = 200;
        private const string StealthFormat = " *Stealth: {0}";
        #endregion

        #region Constructor
        public Fighter(string name, double attackPoints, double defensePoints, bool stealthMode)
        {
            this.Name = name;
            this.AttackPoints = attackPoints;
            this.DefensePoints = defensePoints;
            this.StealthMode = stealthMode;
            this.HealthPoints = InitialHelathPoints;
            this.Targets = new List<string>();
        }
        #endregion

        #region Properies
        public bool StealthMode { get; private set; }
        #endregion

        #region Methods
        public void ToggleStealthMode()
        {
            this.StealthMode = this.StealthMode ? false : true;
        }

        public override string ToString()
        {
            var result = new StringBuilder();
            result.AppendLine(base.ToString());
            result.AppendFormat(StealthFormat, this.StealthMode ? "ON" : "OFF");
            return result.ToString();
        }
        #endregion
    }
}
namespace WarMachines.Machines
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using WarMachines.Interfaces;

    public abstract class Machine : IMachine
    {
        #region Constants
        private const string NameFormat = " - {0}";
        private const string TypeFormat = " *Type: {0}";
        private const string HealthFormat = " *Health: {0}";
        private const string AttackFormat = " *Attack: {0}";
        private const string DefenseFormat = " *Defense: {0}";
        private const string TargetsFormat = " *Targets: {0}";
        private const string NoTargets = "None";
        #endregion

        #region Fields
        private IList<string> targets;
        private IPilot pilot;
        private string name;
        #endregion

        #region Properties
        public string Name
        {
            get
            {
                return string.Copy(this.name);
            }

            set
            {
                if (string.IsNullOrEmpty(value))
                {
                    throw new ArgumentNullException("Machine name cannot be null or empty.");
                }

                this.name = value;
            }
        }

        public IPilot Pilot
        {
            get
            {
                return this.pilot;
            }

            set
            {
                if (value == null)
                {
                    throw new ArgumentNullException("Machine cannot be engaged without pilot.");
                }

                this.pilot = value;
            }
        }

        public double HealthPoints { get; set; }

        public double AttackPoints { get; protected set; }

        public double DefensePoints { get; protected set; }

        public IList<string> Targets
        {
            get
            {
                return this.targets;
            }

            protected set
            {
                if (value == null)
                {
                    throw new ArgumentNullException("Targets cannot be null.");
                }

                this.targets = value;
            }
        }
        #endregion

        #region Public methods
        public void Attack(string target)
        {
            if (string.IsNullOrEmpty(target))
            {
                throw new ArgumentNullException("Target attacked cannot be null.");
            }

            this.targets.Add(target);
        }

        public override string ToString()
        {
            string targetsToStr = this.targets.Count == 0 ? NoTargets : string.Join(", ", this.targets);
            string typeName = this.GetType().Name;
            StringBuilder result = new StringBuilder();
            result.AppendFormat(NameFormat, this.name);
            result.AppendLine();
            result.AppendFormat(TypeFormat, typeName);
            result.AppendLine();
            result.AppendFormat(HealthFormat, this.HealthPoints);
            result.AppendLine();
            result.AppendFormat(AttackFormat, this.AttackPoints);
            result.AppendLine();
            result.AppendFormat(DefenseFormat, this.DefensePoints);
            result.AppendLine();
            result.AppendFormat(TargetsFormat, targetsToStr);
            return result.ToString();
        }
        #endregion
    }
}
namespace WarMachines.Machines
```

```
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using WarMachines.Interfaces;

    public class Pilot : IPilot
    {
        #region Fields
        private const string NoMachines = "no" + Machines;
        private const string FirstLineFormat = "{0} - {1}";
        private const string Machines = " machines";
        private const string OneMachine = "1 machine";

        private string name;
        private IList<IMachine> machines;
        #endregion

        #region Constructor
        public Pilot(string name)
        {
            this.Name = name;
            this.machines = new List<IMachine>();
        }
        #endregion

        #region Properties
        public string Name
        {
            get
            {
                return string.Copy(this.name);
            }

            private set
            {
                if (string.IsNullOrEmpty(value))
                {
                    throw new ArgumentNullException("Pilot should have a name.");
                }

                this.name = value;
            }
        }
        #endregion

        #region Method
        public void AddMachine(IMachine machine)
        {
            if (machines == null)
            {
                throw new ArgumentNullException("Machine cannot be null.");
            }

            this.machines.Add(machine);
        }

        public string Report()
        {
            string numberOfMachines;
            if (this.machines.Count == 0)
            {
                numberOfMachines = NoMachines;
            }
            else if (this.machines.Count == 1)
            {
                numberOfMachines = OneMachine;
            }
            else
            {
                numberOfMachines = this.machines.Count + Machines;
            }
            var result = new StringBuilder();
            result.AppendFormat(FirstLineFormat, this.name, numberOfMachines);
            if (this.machines.Count != 0)
            {
                result.AppendLine();
                var sortedMachines = this.machines.OrderBy(m => m.HealthPoints).ThenBy(m => m.Name).ToList() ↙
    ;

                for (int count = 0; count < sortedMachines.Count; count++)
                {
                    result.Append(sortedMachines[count].ToString());
                    if (count != sortedMachines.Count - 1)
                    {
                        result.AppendLine();
                    }
                }
            }

            return result.ToString();
        }
        #endregion
    }
}
namespace WarMachines.Machines
{
    using System;
    using System.Collections.Generic;
    using System.Linq;
    using System.Text;
    using WarMachines.Interfaces;

    public class Tank : Machine, ITank
    {
        #region Fields
        private const double InitialHealthPoints = 100;
        private const double DefenceModeDefenceBounus = 30;
        private const double DefenceModeAttackMinus = 40;
        private const string DefenseFormat = " *Defense: {0}";
        #endregion

        #region Constant
        public Tank(string name, double attackPoints, double defensePoints)
        {
            this.Name = name;
            this.AttackPoints = attackPoints;
            this.DefensePoints = defensePoints;
            this.HealthPoints = InitialHealthPoints;
            this.DefenseMode = false;
            this.ToggleDefenseMode();
            this.Targets = new List<string>();
        }
        #endregion

        #region Properties
        public bool DefenseMode { get; private set; }
        #endregion

        #region Methods
        public void ToggleDefenseMode()
        {
            if (this.DefenseMode)
            {
                this.DefenseMode = false;
                this.AttackPoints += DefenceModeAttackMinus;
                this.DefensePoints -= DefenceModeDefenceBounus;
            }
            else
            {
                this.DefenseMode = true;
                this.AttackPoints -= DefenceModeAttackMinus;
                this.DefensePoints += DefenceModeDefenceBounus;
            }
        }
        public override string ToString()
        {
            StringBuilder result = new StringBuilder();
            result.AppendLine(base.ToString());
            result.AppendFormat(DefenseFormat, this.DefenseMode ? "ON" : "OFF");
            return result.ToString();
        }
        #endregion
    }
}
namespace WarMachines
{
    using WarMachines.Engine;

    public class WarMachinesProgram
    {
        public static void Main()
        {
            WarMachineEngine.Instance.Start();
        }
    }
}
```