1) Donnez le code C# complet des classes Wheel, Trailer et Driver.

```
using System;

namespace TP_07
{
    public class Trailer
    {
        #region attributs
        // Size of the trailer
        private int size;
        // Car the trailer belongs to
        private Car car;
        #endregion

        #region accesseurs
        /// <summary>
        /// Gets or sets the car the trailer belongs to.
        /// </summary>
        public Car Car
        {
            get => car;
            set => car = value;
        }
        #endregion

        #region constructeurs
        /// <summary>
        /// Initializes a new instance of the Trailer class with a specified size.
        /// </summary>
        /// <param name="taille">The size of the trailer.</param>
        public Trailer(int taille)
        {
            this.size = taille;
        }

        /// <summary>
        /// Initializes a new instance of the Trailer class that is a copy of the current instance.
        /// </summary>
        /// <param name="trailer">The trailer to copy.</param>
        public Trailer(Trailer trailer)
        {
            this.size = trailer.size;
        }
        #endregion

        #region méthodes
        /// <summary>
        /// Returns a string that represents the current trailer.
        /// </summary>
        /// <returns>A string that represents the current trailer.</returns>
        public override string ToString()
        {
            return "[Trailer] capacity "+this.size+"l";
        }

        /// <summary>
        /// Determines whether the specified Trailer is equal to the current Trailer.
        /// </summary>
        /// <param name="other">The Trailer to compare with the current Trailer.</param>
        /// <returns>true if the specified Trailer is equal to the current Trailer; otherwise, false.</returns>
        protected bool Equals(Trailer other)
        {
            return size == other.size;
        }

        /// <summary>
        /// Determines whether the specified object is equal to the current Trailer.
        /// </summary>
        /// <param name="obj">The object to compare with the current Trailer.</param>
        /// <returns>true if the specified object is equal to the current Trailer; otherwise, false.</returns>
        public override bool Equals(object? obj)
        {
            if (ReferenceEquals(null, obj)) return false;
            if (ReferenceEquals(this, obj)) return true;
            if (obj.GetType() != this.GetType()) return false;
            return Equals((Trailer) obj);
        }

        /// <summary>
        /// Serves as a hash function for a Trailer object.
        /// </summary>
        /// <returns>A hash code for this instance that is suitable for use in hashing algorithms and data structures such as a hash table.</returns>
        public override int GetHashCode()
        {
            return size;
        }
        #endregion
    }
}
```

```cs
using System;

namespace TP_07
{
    public class Wheel
    {
        #region attributs
        // Size of the wheel
        private int size;
        // Car the wheel belongs to
        private Car car;
        #endregion

        #region accesseurs
        /// <summary>
        /// Gets or sets the car the wheel belongs to.
        /// </summary>
        public Car Car
        {
            get => car;
            set => car = value;
        }
        #endregion

        #region constructeurs
        /// <summary>
        /// Initializes a new instance of the Wheel class with a specified size.
        /// </summary>
        /// <param name="taille">The size of the wheel.</param>
        public Wheel(int taille)
        {
            this.size = taille;
        }

        /// <summary>
        /// Initializes a new instance of the Wheel class that is a copy of the current instance.
        /// </summary>
        /// <param name="wheel">The wheel to copy.</param>
        public Wheel(Wheel wheel)
        {
            this.size = wheel.size;
        }
        #endregion

        #region méthodes
        /// <summary>
        /// Returns a string that represents the current wheel.
        /// </summary>
        /// <returns>A string that represents the current wheel.</returns>
        public override string ToString()
        {
            return "[Wheel] " + this.size.ToString() + " inches";
        }

        /// <summary>
        /// Determines whether the specified Wheel is equal to the current Wheel.
        /// </summary>
        /// <param name="other">The Wheel to compare with the current Wheel.</param>
        /// <returns>true if the specified Wheel is equal to the current Wheel; otherwise, false.</returns>
        protected bool Equals(Wheel other)
        {
            return size == other.size;
        }

        /// <summary>
        /// Determines whether the specified object is equal to the current Wheel.
        /// </summary>
        /// <param name="obj">The object to compare with the current Wheel.</param>
        /// <returns>true if the specified object is equal to the current Wheel; otherwise, false.</returns>
        public override bool Equals(object? obj)
        {
            if (ReferenceEquals(null, obj)) return false;
            if (ReferenceEquals(this, obj)) return true;
            if (obj.GetType() != this.GetType()) return false;
            return Equals((Wheel) obj);
        }

        /// <summary>
        /// Serves as a hash function for a Wheel object.
        /// </summary>
        /// <returns>A hash code for this instance that is suitable for use in hashing algorithms and data structures such as a hash table.</returns>
        public override int GetHashCode()
        {
            return size;
        }
        #endregion
    }
}
```

```
using System;

namespace TP_07
{
    public class Driver
    {
        #region attributs
        // Name of the driver
        private string name;
        // Car the driver is driving
        private Car car;
        #endregion

        #region accesseurs
        /// <summary>
        /// Gets or sets the car the driver is driving.
        /// </summary>
        public Car Car
        {
            get => car;
            set => car = value;
        }
        #endregion

        #region constructeurs
        /// <summary>
        /// Initializes a new instance of the Driver class with a specified name.
        /// </summary>
        /// <param name="nom">The name of the driver.</param>
        public Driver(string nom)
        {
            this.name = nom;
        }

        /// <summary>
        /// Initializes a new instance of the Driver class that is a copy of the current instance.
        /// </summary>
        /// <param name="conducteur">The driver to copy.</param>
        public Driver(Driver conducteur)
        {
            this.name = conducteur.name;
        }
        #endregion

        #region méthodes
        /// <summary>
        /// Returns a string that represents the current driver.
        /// </summary>
        /// <returns>A string that represents the current driver.</returns>
        public override string ToString()
        {
            return "[Driver] "+this.name;
        }

        /// <summary>
        /// Determines whether the specified Driver is equal to the current Driver.
        /// </summary>
        /// <param name="other">The Driver to compare with the current Driver.</param>
        /// <returns>true if the specified Driver is equal to the current Driver; otherwise, false.</returns>
        protected bool Equals(Driver other)
        {
            return name == other.name;
        }

        /// <summary>
        /// Determines whether the specified object is equal to the current Driver.
        /// </summary>
        /// <param name="obj">The object to compare with the current Driver.</param>
        /// <returns>true if the specified object is equal to the current Driver; otherwise, false.</returns>
        public override bool Equals(object? obj)
        {
            if (ReferenceEquals(null, obj)) return false;
            if (ReferenceEquals(this, obj)) return true;
            if (obj.GetType() != this.GetType()) return false;
            return Equals((Driver) obj);
        }

        /// <summary>
        /// Serves as a hash function for a Driver object.
        /// </summary>
        /// <returns>A hash code for this instance that is suitable for use in hashing algorithms and data structures such as a hash table.</returns>
        public override int GetHashCode()
        {
            return name.GetHashCode();
        }
        #endregion
    }
}
```

2) Expliquez pourquoi le constructeur de Engine prend un Car en paramètre

Le constructeur de la classe Engine reçoit en paramètre un objet de type Car afin de créer une liaison entre l'instance de Engine (le moteur) et celle de Car (la voiture). Ainsi, chaque Engine est lié à une Car particulière.

3) Donnez le code source de la classe Engine

```
using System;

namespace TP_07
{
    public class Engine
    {
        #region attributs
        // Power of the engine
        private int power;
        // Is the engine running
        private bool isRunning;
        // Has the engine failed
        private bool failure;
        // Car the engine belongs to
        private Car car;
        #endregion

        #region accesseurs
        /// <summary>
        /// Gets or sets the power of the engine.
        /// </summary>
        public int Power
        {
            get => power;
            set => power = value;
        }

        /// <summary>
        /// Gets or sets whether the engine is running.
        /// </summary>
        public bool IsRunning
        {
            get => isRunning;
            set => isRunning = value;
        }

        /// <summary>
        /// Gets or sets whether the engine has failed.
        /// </summary>
        public bool Failure
        {
            get => failure;
            set => failure = value;
        }

        /// <summary>
        /// Gets or sets the car the engine belongs to.
        /// </summary>
        public Car Car
        {
            get => car;
            set => car = value;
        }
        #endregion

        #region constructeurs
        /// <summary>
        /// Initializes a new instance of the Engine class with a specified power and car.
        /// </summary>
        /// <param name="pouvoir">The power of the engine.</param>
        /// <param name="voiture">The car the engine belongs to.</param>
        public Engine(int pouvoir, Car voiture)
        {
            this.power = pouvoir;
            this.car = voiture;
            this.isRunning = false;
            this.failure = false;
        }

        /// <summary>
        /// Initializes a new instance of the Engine class that is a copy of the current instance.
        /// </summary>
        /// <param name="engine">The engine to copy.</param>
        public Engine(Engine engine)
        {
            this.power = engine.power;
            this.car = engine.car;
            this.isRunning = engine.isRunning;
            this.failure = engine.failure;
        }
        #endregion
```

```
        #region méthodes
        /// <summary>
        /// Starts the engine.
        /// </summary>
        public void Start()
        {
            this.isRunning = true;
        }

        /// <summary>
        /// Stops the engine.
        /// </summary>
        public void Stop()
        {
            this.isRunning = false;
        }

        /// <summary>
        /// Fails the engine.
        /// </summary>
        public void Fail()
        {
            this.failure = true;
            this.isRunning = false;
        }

        /// <summary>
        /// Repairs the engine.
        /// </summary>
        public void Repair()
        {
            this.failure = false;
        }

        /// <summary>
        /// Returns a string that represents the current engine.
        /// </summary>
        /// <returns>A string that represents the current engine.</returns>
        public override string ToString()
        {
            if (this.isRunning)
            {
                return "[Engine] power "+this.power+", running";
            }
            else
            {
                if (failure)
                {
                    return "[Engine] power "+this.power+", failure";
                }
                else
                {
                    return "[Engine] power "+this.power+", stopped";
                }
            }
        }
        #endregion
    }
}
```

4) Expliquez les paramètres du constructeur de Car

Le constructeur de la classe `Car` accepte quatre arguments : `pouvoir`, `modele`, `nombRoue`, et `tailleRoue`.

1. `pouvoir` : Il s'agit d'un entier définissant la puissance du moteur de la voiture. Cette donnée sert à créer une instance de la classe `Engine` qui sera attribuée à la voiture.

2. `modele` : Ce paramètre est une chaîne de caractères identifiant le modèle de la voiture. Il est affecté à l'attribut `model` de l'instance de la voiture.

3. `nombRoue` : Un entier indiquant le nombre de roues de la voiture. À partir de cette valeur, un tableau de roues (`Wheel[]`) est constitué, sa taille correspondant au nombre indiqué. Chaque roue du tableau est par la suite initialisée en utilisant la dimension donnée par `tailleRoue`.

4. `tailleRoue` : Un entier spécifiant la dimension de chaque roue sur la voiture. Cette information est employée pour définir la taille de chaque roue dans le tableau de roues.

5) Donnez le code complet de Car

```
namespace TP_07
{
    public class Car
    {
        #region attributs
        // Model of the car
        private string model;
        // Engine of the car
        private Engine engine;
        // Array of wheels of the car
        private Wheel[] wheels;
        // Driver of the car
        private Driver? driver;
        // Trailer attached to the car
        private Trailer? trailer;
        #endregion

        #region constructeurs
        /// <summary>
        /// Initializes a new instance of the Car class with specified power, model, number of wheels, and wheel size.
        /// </summary>
        public Car(int pouvoir, string modele, int nombRoue, int tailleRoue)
        {
            this.model = modele;
            this.engine = new Engine(pouvoir, this);

            this.wheels = new Wheel[nombRoue];
            for (int i = 0; i < nombRoue; i++)
            {
                this.wheels[i] = new Wheel(tailleRoue);
            }
        }

        /// <summary>
        /// Initializes a new instance of the Car class that is a copy of the current instance.
        /// </summary>
        public Car(Car voiture) {
            this.model = voiture.model;
            this.engine = new Engine(voiture.engine);
            this.wheels = new Wheel[voiture.wheels.Length];
            for (int i = 0; i < voiture.wheels.Length; i++)
            {
                this.wheels[i] = new Wheel(voiture.wheels[i]);
            }
        }
        #endregion

        #region méthodes
        /// <summary>
        /// Fails the engine of the car.
        /// </summary>
        public void Fail()
        {
            this.engine.Fail();
        }

        /// <summary>
        /// Repairs the engine of the car.
        /// </summary>
        public void Repair()
        {
            this.engine.Repair();
        }

        /// <summary>
        /// Starts the engine of the car.
        /// </summary>
        public void Start()
        {
            this.engine.Start();
        }

        /// <summary>
        /// Stops the engine of the car.
        /// </summary>
        public void Stop()
        {
            this.engine.Stop();
        }

        /// <summary>
        /// Changes the driver of the car.
        /// </summary>
        public void ChangeDriver(Driver conducteur)
        {
            this.driver = conducteur;
        }

        /// <summary>
        /// Attaches a trailer to the car.
        /// </summary>
        public void AttachTrailer(Trailer trail)
        {
            this.trailer = trail;
        }
```

```
    /// <summary>
    /// Unties the trailer from the car.
    /// </summary>
    public void UntieTrailler()
    {
        this.trailer = null;
    }

    /// <summary>
    /// Returns a string that represents the current car.
    /// </summary>
    public override string ToString()
    {
        string chaine = "[Car] " + this.model + "\n";

        chaine += "->" + this.engine.ToString() + "\n";

        foreach (var roue  in this.wheels)
        {
            chaine += "->" + roue.ToString() + " ";
        }
        chaine += "\n";

        if (this.driver != null)
        {
            chaine += "->" + this.driver.ToString() + "\n";
        }

        if (this.trailer != null)
        {
            chaine += "->" + this.trailer.ToString() + "\n";
        }

        return chaine;
    }
    #endregion
  }
}
```

6) Donnez le code des constructeurs par copie des classes Driver et Trailer

```
/// <summary>
/// Initializes a new instance of the Driver class that is a copy of the current instance.
/// </summary>
/// <param name="conducteur">The driver to copy.</param>
public Driver(Driver conducteur)
{
    this.name = conducteur.name;
    this.car = conducteur.car;
}
```

```
/// <summary>
/// Initializes a new instance of the Trailer class that is a copy of the current instance.
/// </summary>
/// <param name="trailer">The trailer to copy.</param>
public Trailer(Trailer trailer)
{
    this.size = trailer.size;
    this.car = trailer.car;
}
```

7) Indiquez, pour la classe Car, la nature de chaque association

Pour la classe Car, les associations se définissent comme suit :

1. `Engine engineVar` : Cette relation est de type composition. Le moteur (Engine) est essentiellement lié à la voiture (Car), avec son cycle de vie dépendant entièrement de celui de la voiture. Ainsi, lors de la duplication d'une voiture, un nouveau moteur est systématiquement créé pour accompagner la nouvelle voiture.

2. `Wheel[] wheelsVar` : C'est également une composition. Les roues (Wheel) sont fondamentalement attachées à la voiture, leur existence étant intimement liée à celle de cette dernière. Par conséquent, lorsqu'une voiture est dupliquée, il est nécessaire de créer de nouvelles instances des roues pour équiper la nouvelle voiture.

3. `Driver driverVar` : Ici, nous avons affaire à une association simple. Le conducteur (Driver) n'est pas dépendant de la voiture en termes de cycle de vie. De ce fait, lors de la duplication d'une voiture, le conducteur initial peut être associé à la nouvelle voiture sans nécessiter de création d'une nouvelle instance de conducteur.

4. `Trailer trailerVar` : Cette relation est aussi une association simple. Une remorque (Trailer) n'est pas conditionnée par le cycle de vie de la voiture à laquelle elle peut être attelée. Ainsi, lors de la copie d'une voiture, la nouvelle voiture peut être associée à la même remorque existante.

8) Donnez le code du constructeur par copie de la classe Car

```
/// <summary>
/// Initializes a new instance of the Car class that is a copy of the current instance.
/// </summary>
public Car(Car voiture)
{
    this.model = voiture.model;
    this.engine = new Engine(voiture.engine);
    this.wheels = voiture.wheels;
    this.driver = voiture.driver;
    this.trailer = voiture.trailer;
}
```

9) Expliquez pourquoi le constructeur par copie de Engine ne peut pas avoir qu'un seul paramètre

Le constructeur de copie de la classe Engine nécessite deux arguments : une instance de Engine à dupliquer et une instance de Car à laquelle le nouveau moteur sera associé. Cette exigence découle directement du type d'association existant entre Engine et Car. Sans la spécification d'une instance de Car, il serait impossible de déterminer avec quelle voiture la nouvelle instance de Engine devrait être liée.

10) Donnez le code du constructeur par copie de la classe Engine

```
/// <summary>
/// Initializes a new instance of the Engine class by copying an existing instance.
/// </summary>
/// <param name="engine">The engine to copy.</param>
public Engine(Engine engine) {
    this.power = engine.power;
    this.car = engine.car;
    this.isRunning = engine.isRunning;
    this.failure = engine.failure;
}
```

11) Est-ce qu'il n'y a pas un problème ? Pouvez-vous réaliser proprement Engine.Equals ? Pourquoi ?

La classe Engine présente un souci au niveau de sa méthode Equals, qui vise à déterminer si deux instances de Engine sont identiques. Toutefois, elle néglige de prendre en compte la référence à une voiture, ce qui peut conduire à des erreurs. Pour corriger cela, la solution consiste à comparer uniquement les attributs propres à Engine qui ne concernent pas des associations, afin d'éviter les complications. Voici une approche pour réaliser cette comparaison de manière adéquate.

```
/// <summary>
/// Determines whether the specified Engine is equal to the current Engine.
/// </summary>
/// <param name="other">The Engine to compare with the current Engine.</param>
/// <returns>true if the specified Engine is equal to the current Engine; otherwise, false.</returns>
public override bool Equals(object obj)
{
    if (obj == null || GetType() != obj.GetType())
    {
        return false;
    }

    Engine engine = (Engine)obj;
    return power == engine.power && isRunning == engine.isRunning && failure == engine.failure;
}
```

12) Que pensez-vous du fait que la composition Car/Engine soit bidirectionnelle ? Est-ce utile ? nécessaire ? qu'est-ce qui serait possible dans le cas contraire ?

La composition bidirectionnelle entre Car et Engine, où Car possède une référence à Engine et Engine a une référence à Car, peut s'avérer utile dans certaines situations. Par exemple, cela permet de naviguer facilement de Engine à Car, ce qui est pratique si l'on souhaite identifier à quelle voiture un moteur spécifique est associé.

Néanmoins, cette configuration peut aussi complexifier le code. Par exemple, elle peut compliquer l'implémentation de méthodes telles que Equals, du fait de la nécessité d'éviter les boucles de références infinies. De plus, cela peut augmenter la difficulté de compréhension et de maintenance du code, étant donné qu'il est essentiel de veiller à ce que les relations entre Car et Engine soient correctement gérées à chaque création, modification, ou suppression d'instances.

Si la composition n'était pas bidirectionnelle, c'est-à-dire si seulement Car référençait Engine sans que le contraire soit vrai, le code en ressortirait simplifié. Dans un tel cas, l'implémentation de méthodes comme Equals deviendrait plus aisée, car il n'y aurait pas à craindre la création de boucles de référence infinies. Toutefois, cette simplification aurait pour inconvénient de perdre la capacité de naviguer de Engine à Car, ce qui pourrait ne pas convenir selon les besoins spécifiques de votre application.