**Лабораторна робота оцінюється в 2 бали плюс 1 бал за додаткове завдання.**

**Короткі теоретичні відомості**

**Процес фіналізації(**[**Garbage Collection**](https://docs.microsoft.com/ru-ru/dotnet/standard/garbage-collection/index)**).**

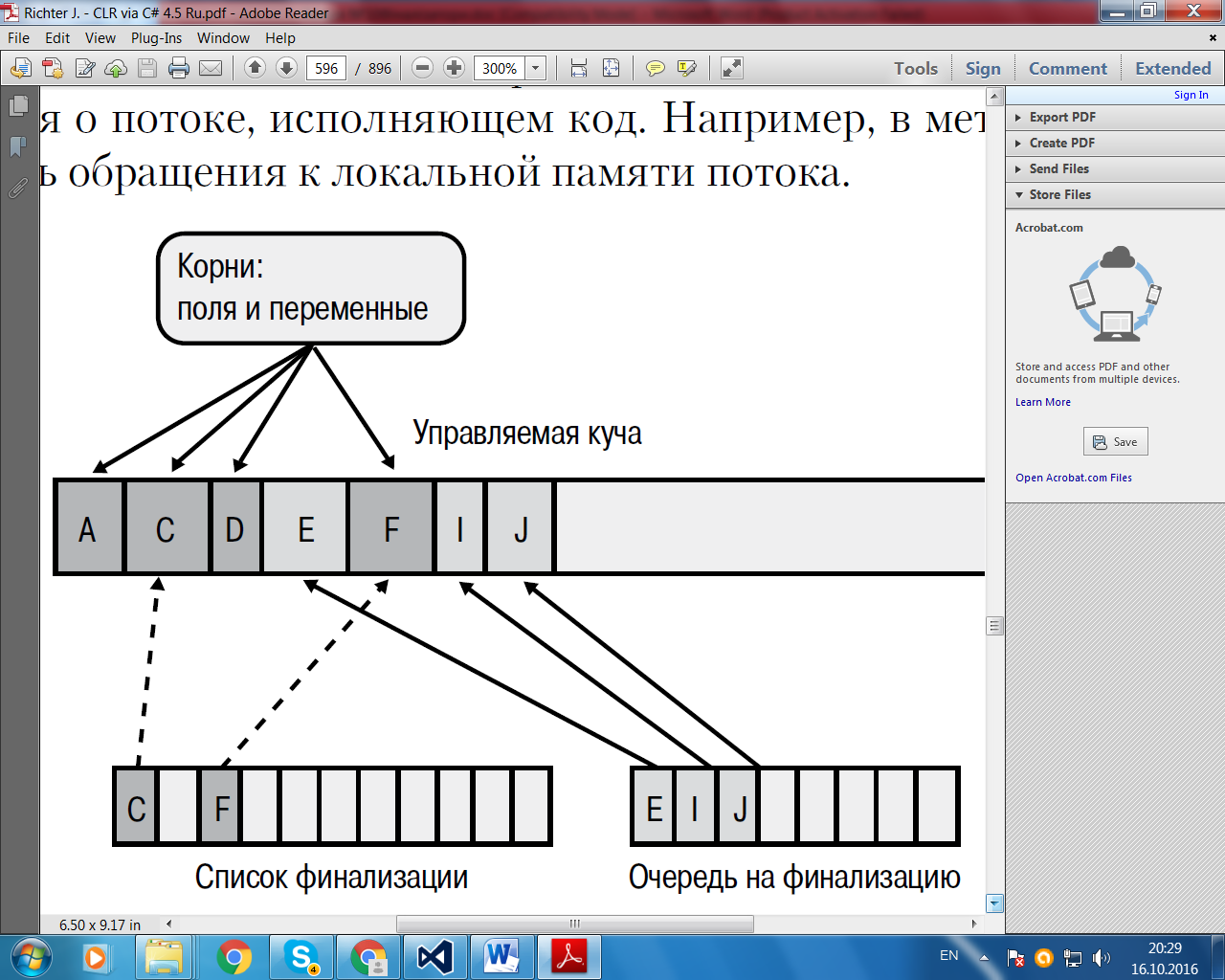
**Збирач сміття.**

Коли для виділення пам‘яті під наступний об‘єкт CLR з‘ясовує, що вільної пам‘яті в керованій кучі недостатньо, в окремому процесі запускається збирач сміття(GC). Він призупиняє всі активні потоки в рамках поточного процесу, щоб виключити доступ до кучі до закінчення роботи збирача сміття. Після закінчення роботи збирача сміття потоки продовжують роботу.

Як же збирач сміття визначає, які об‘єкти можна видалити? Якщо на даний об‘єкт немає щодного посилання, то об‘єкт видаляється, а вільний простір в кучі стискається до компактного стану.

**Фіналізатори**.

Середовище CLR відслідковує кожне посилання на керуємий об’єкт в системі. Якщо об’єкт стає недоступним ні по одному із послань, він помічається для видалення. Збирач сміття або звільняє пам'ять, яку займає об’єкт, або ставить в чергу на видалення, якщо у об’єкта є фіналізатор. За роботу фіналізатора відповідає окремий потік CLR – потік фіналізатора. Він проходить по черзі об’єктів і викликає їх фіналізатори. Після завершення цього процесу при наступному проході збирача сміття об’єкт остаточно видаляється, а пам'ять звільняється.



Розглянуте знищення об’єктів називається недетермінованим, бо не можна сказати в який саме час воно буде виконано. Такий фіналізатор треба створювати тільки якщо використовуються **некеровані ресурси**(керовані ресурси таким чином звільняти не можна).

В С++ деструктор викликався в відомий час. Таке знищення об’єкта називається детермінованим. В C# для виконання такого знищення використовується інтерфейс IDisposable. Він містить єдиний метод Dispose(), який повинен виконати всю роботу по звільненню ресурсів. В цьому методі можна звільняти як **керовані, так і некеровані ресурси**. Недолік – цей метод викликається програмістом. При реалізації цього методу клас звичайно будується так, що якщо програміст не викликав Dispose, його буде викликано фінілізатором.

class Base : IDisposable {

bool b = false;

~Base() {

Console.WriteLine(“dis Base”);

if (!b) Dispose();

}

public void Dispose(){

b = true;

Console.WriteLine(“Disp”);

GC.SuppressFinalize(this);

}

public void Print() {

Console.WriteLine(“Print”);

}

}

static void Main(string[] args) {

Base b = new Base();

try { b.Print();}

finally{

b.Dispose();

}

Console.Read();}

Результат:

Print

Disp

Виклик методу GC.SuppressFinalize -запобігає фіналізації об’єкту.

**Використання ключового слова using.**

Ключове слово using було перевантажено для підтримки шаблона Disposable.

Приклад використання:

static void Main(string[] args){

using( Base b = new Base()){ b.Print();}

Console.Read();

}

В коді MSIL using буде представлено блоком try/finally, де в блоці finally буде викликано метод Dispose. Такий запис можливий, тільки якщо клас Base реалізує інтерфейс IDisposable

Завдання: Продемонструйте в коді:

1. Використання методу Dispose та інтерфейсу IDisposable.
2. Використання методу Finalize.
3. Використання ключового слова using при фіналізації об‘єктів ( **для демонстрації обрати один із бібліотечних класів, який реалізує вказаний інтерфейс**).
4. Роботу з трьома методами GC.
5. Додатково:
   1. реалізація паттерну Dispose.

Контрольні питання:

1. Поняття поколінь при роботі GC.
2. Чому треба уникати перевизначення методу Finalize.
3. Інтерфейс IDisposable.
4. Призначення деструктора C#.
5. Детерміноване та недетерміноване звільнення ресурсів.
6. Призначення ключового слова using