AD SOYAD:	**TESLİM TARİHİ:**	
OKUL NO:	**TESLİM SÜRESİ: 2 hafta**	**ÖDEV NO: 3**

1-

- **Life Game Java konsol ortamında 17x17 lik bir alanda (hücre habitatı (yaşam alanı)) oynanacaktır. Fakat `hucreHabitat[][]` dizisi 19x19 olarak belirlenmiş ve şekil 2 de görüldüğü gibi 0. ve 18. satır ile 0. ve 18. sütunlar habitat alanına alınmamıştır. Bunun sebebi ise şekil 2 de kırmızı hücre ve komşularını gösterilen 1,1 hücresinde görüldüğü gibi eğer 1,1 yerine 0,0 hücresi habitat içerisinde olsaydı komşu hücreleri bulunmayacağı için yazılımda dizi sınırı taşma problemleri ile karşılaşıla bilinir.**
- **Life Game için her iterasyonda ekran çıktısı karakterler ile gösterilecektir.**

LIFE GAME KURALLARI:

Kısaca "LIFE" diye de anılan "LIFE GAME" İngiliz matematikçi John Horton Conway tarafından 1970 yılında hücresel simülasyon amaçlı olarak tasarlanmış bir hücresel otomattır. (Cellular Automata, Cellular Spaces)

"Game" oyuncusuz bir oyundur. Bunun anlamı temel bir başlangıç durumundan başlayarak, gelişimin iterasyonlar boyunca devam etmesidir. Yani hücrelerin "hayatta kalma" ve "ölüm" kurallarına göre dizayn edimiş bir simülasyon aracı olarak tasarlanmıştır.

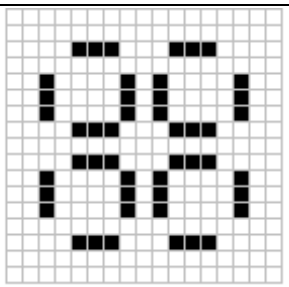
Bu simülasyonda hücrelerin "YAŞAM" ve "ÖLÜM" olmak üzere iki durumu söz konusudur.

Bu oyunda amaç belli bir başlangıç durumu referans alınarak, belli bir bölge içerisinde, herbir yaşam döngüsünde (itrasyonda) hücrelerin hayat ve ölüm durumlarını test edip sonuçları görmektir.

LIFE GAME' nin evreni, her birinin sadece "CANLI", "ÖLÜ" olmak üzere 2 olası duruma sahip olduğu 2 boyutlu karelerden oluşur. Her bir kare bir hücreyi temsil eder. Her hücre yatayda dikeyde ve çaprazda olmak üzere 8 bitişik komşu hücre (şekil-2 kırmızı hücre ve komşuları) ile etkileşim içindedir. Her zaman iterasyonunda aşağıdaki kurallara göre bir sonraki durum belli olur:

1- **Eğer canlı bir hücrenin 2 canlı hücre hücreden az komşusu olursa o hücre ölür.**
2- **Eğer canlı bir hücrenin 2 veya 3 canlı hücre komşusu olursa o hücre bir sonraki nesilde yaşar.**
3- **Eğer canlı bir hücrenin 3 den fazla canlı hücre komşusu olursa o hücre bir sonraki nesilde ölür.**
4- **Eğer ölü bir hücrenin 3 canlı hücre komşusu olursa o hücre bir sonraki nesilde yeniden üretilir ve yaşamaya başlar.**

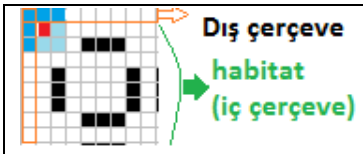
LIFE GAME evreninin başlangıç deseni sistemin çekirdeğini oluşturur. İlk nesil yukarıdaki kuralları uygulayarak aynı anda çekirdekteki tüm hücrelere aynı anda yaratılır. Ölüm ve yaşam aynı anda meydan gelir. Her bir değişim döngüsüne bir **iterasyon** denir. Simülasyon yukarıdaki kurallar dahilinde devam eder.



Şekil-1: Pulsar deseni

TAVSİYE EDİLEN JAVA YORDAMLARI:

Uygulamada bir sınıf ve birden fazla yordam yazılması olması tavsiye edilir. Yazılması istenen yordamlar şekil-4'de gösterilmiş ve aşağıda açıklanarak örnek kod alanı verilmiştir.



Şekil-2: komşu hücre, iç ve dış çerçeve

```
----------------
----------------
----###---###----
----------------
--#----#-#----#--
--#----#-#----#--
--#----#-#----#--
----###---###----
----------------
----###---###----
--#----#-#----#--
--#----#-#----#--
--#----#-#----#--
----------------
----###---###----
----------------
----------------
```

(a)

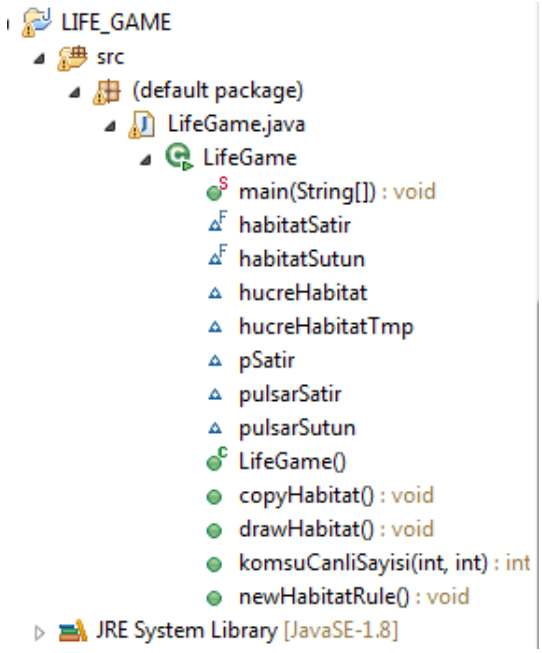
```
----------------
-----#-----#-----
-----#-----#-----
-----##---##-----
----------------
-###--##-##--###-
---#-#-#-#-#-#---
-----##---##-----
----------------
-----##---##-----
---#-#-#-#-#-#---
-###--##-##--###-
----------------
-----##---##-----
-----#-----#-----
-----#-----#-----
----------------
```

(b)

```
----------------
----------------
----##-----##----
-----##---##-----
--#--#-#-#-#--#--
--###-##-##-###--
---#-#-#-#-#-#---
----###---###----
----------------
----###---###----
---#-#-#-#-#-#---
--###-##-##-###--
--#--#-#-#-#--#--
-----##---##-----
----##-----##----
----------------
----------------
```

(c)

Şekil-3: (a) java konsol ortamı pulsar başlangıç deseni, (b) birinci iterasyon, (c) ikinci iterasyon



Şekil-4: LIFE_GAME sınıfı için tavsiye edilen Java yordamları ve değişkenleri

1. public LifeGame():

Life Game yapılandırıcı yordamıdır. 19 x 19 lik bir habitat üzerinde simüle edilecektir. Fakat bu habitatın içdeki 17 x 17 lik bölümü gerçek yaçam alanı olacaktır. Simülasyon başlamadan önce habitat şekil-1 de verilen pulsar deseni ile başlamalıdır. Bu nedenle gerçek habitat ve geçici habitat sıfırlanmalı, pulsar deseni habitata yüklenmelidir. Yapılandırıcı için gerekli java kodları aşağıda verilmiştir.

2. public void drawHabitat():

Gerçek yaşam alanı (`hucreHabitat[][]`) şekil-3'de gösterildiği formatla konsol ekranına bastıran yordamdır. Burada ölü hücreler için tire (-), canlı hücreler için diyez (#) karakterleri tercih edilmiştir. Şekil-2'de gösterildiği gibi habitat olarak dış çerçeve değil iç çerçeve bastırılmalıdır.

3. public int komsuCanliSayisi(int satir, int sutun):

Koordinatları satır sütun şeklinde parametre olarak girilen hücre merkezde olmak üzere etrafındaki 3x3 lük alanda canlı komşu sayımı yapılacaktır. Eğer kendisi de canlı ise, canlı komşu sayısına eklenmemelidir. Bulunan canlı komşu sayısı geri döndürülür.

4- public void newHabitatRule():

Life Game' in yukarıda sıralanan 4 kuralına göre gerçek habitata bakılarak bir sonraki iterasyondaki hücre nesli için geçici habitat (hucreHabitatTmp) güncellenir. Bunun için sırayla iç çerçevede kalan her bir hücrenin canlı komşu sayısı ve hücrenin canlı olup olmadığına bakılarak hucreHabitatTmp güncellenir. Bu güncellemeden sonra ise yedek habitatın gerçek habitata kopyalama işlemi yapılır.

5- public void copyHabitat():

hucreHabitatTmp'de değerler gerçek yaşam alanına aktarılır.

LIFE GAME JAVA ŞABLON KODLARI:

```
import java.io.IOException;
public class LifeGame {

	final int habitatSatir = 19;
	final int habitatSutun = 19;

	int hucreHabitat[][];
	int hucreHabitatTmp[][];
	int[] pulsarSatir;
	int[] pSatir;

	int pulsarSutun[];

	public LifeGame() {
		// pulsar desni oluşması için gerekli ön tanımlamalar
		pSatir = new int[] { 3, 8, 10, 15 };
		pulsarSatir = new int[] { 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0 };
		pulsarSutun = new int[] { 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0 };

		// gerçek yaşam alanı ve değişikliklerin yapılacağı yedek yaşam alanı tanımı
		hucreHabitat = new int[habitatSatir][habitatSutun];
		hucreHabitatTmp = new int[habitatSatir][habitatSutun];

		// tüm yedek ve gerçek yaşam alanı sıfırlanıyor
		int c = 0;
		for (int i = 0; i < habitatSatir; i++) {
			for (int y = 0; y < habitatSutun; y++) {
				hucreHabitatTmp[i][y] = c;
				hucreHabitat[i][y] = c;
			}
		}
		// pulsar deseni gerçek yaşam alanına atanıyor
		for (int satir = 0; satir < pSatir.length; satir++) {
			for (int sutun = 0; sutun < habitatSutun; sutun++) {
				hucreHabitat[pSatir[satir]][sutun] = pulsarSatir[sutun];
			}
		}

		for (int sutun = 0; sutun < pSatir.length; sutun++) {
			for (int satir = 0; satir < habitatSutun; satir++) {
				hucreHabitat[satir][pSatir[sutun]] = pulsarSatir[satir];
			}
		}
	}

	public void drawHabitat() {
		// gerçek yaşam alanı (hucreHabitat) ekrana çizdiriliyor
		// *******KODLANACAK*********

	}

	public int komsuCanliSayisi(int satir, int sutun) {
		int canliKomsuSayisi = 0;
		// koordinatları girilen hücre merkezde olmak üzere 3x3 lük alanda
		// canlı komşu sayısı tespiti yapılıyor. Eğer kendiside canlı ise
		// canlı komşuya eklenmemelidir.
		// *******KODLANACAK*********
		return canliKomsuSayisi;
	}

	public void newHabitatRule() {
		int cks;
		// Life Game'in 4 kuralına göre gerçek habitata bakılarak
		// bir sonraki iterasyon için geçici habitat (hucreHabitatTmp)
		// güncelleniyor
		// *******KODLANACAK*********
	}

	public void copyHabitat() {
		// yedek hücreden tekrar orjinaline yükleme yap
		// *******KODLANACAK*********
	}

	public static void main(String[] args) throws Exception {
		LifeGame lg = new LifeGame();
		for (int i = 0; i < 20; i++) {
			lg.drawHabitat();
			lg.newHabitatRule();
			System.out.println();
			Thread.sleep(1500);
		}}}
```

2- JAVA'da bağlı liste yapısı kullanılarak bir Yığın uygulaması yazılacaktır. Bu yığın bağlı liste üzerinde çalışacaktır. Bilindiği gibi yığın mantığına göre yığına eklemeler en üste yapılırken, yığından çekmeler ise yine en üstten yapılacaktır. Aşağıda hangi sınıfların yazılacağı ve sınıflarda yazılması gereken metotların nasıl çalışması istendiği anlatılmıştır.

SINIFLAR ve METOTLARI:

1-`public class Eleman:` Bu sınıfta bağlı liste sınıfında verileri taşıyan ve kendi türünden başka bir sınıfa bağlanmayı sağlayan özellikler tanımlanacaktır. Bu uygulamada eleman sınıfında bir kişiye ait ad, soyad, telefon numarası ve doğum tarihi bilgileri tutulacaktır. Bu sınıfın altında yapılandırıcı dışında bir metoda ihtiyaç yoktur.

2-`public class BagliListeYigin` : bu sınıf altında yığın bağlı listesinin başlangıç elemanını tutan bir özellik olmalıdır. Bu sınıf altında yığına ekleme, yığından çekme, yığın yazdırma, yığındaki eleman sayısını döndürme metotları yazılacaktır.

3-`public class AnaSinif` : bu sınıf altında main metodu olacak ve çalışacak ana sınıf olacaktır. Yığına yeni eleman ekleme, yığından silme yığın yazdırma ve yığındaki eleman sayısını gösteren metotların testleri yapılacaktır.

```
public class Eleman {
      String isim;
      String soyisim;
      int telno;
      int dt;

      Eleman(String isim, String
soyisim, int telno, int dt){
            this.isim=isim;
            this.soyisim=soyisim;
            this.telno=telno;
            this.dt=dt;
      }
}
```

3- JAVA'da bağlı liste yapısı kullanılarak bir Kuyruk uygulaması yazılacaktır. Bu kuyruk bağlı liste üzerinde çalışacaktır. Bilindiği gibi kuyruk mantığına göre kuyruğa eklemeler en sona yapılırken, kuyrukdan çekmeler ise yine en önden yapılacaktır. Aşağıda hangi sınıfların yazılacağı ve sınıflarda yazılması gereken metotların nasıl çalışması istendiği anlatılmıştır.

SINIFLAR ve METOTLARI:

1-`public class Eleman:` Bu sınıfta bağlı liste sınıfında verileri taşıyan ve kendi türünden başka bir sınıfa bağlanmayı sağlayan özellikler tanımlanacaktır. Bu uygulamada eleman sınıfında bir kişiye ait ad, soyad, telefon numarası ve doğum tarihi bilgileri tutulacaktır. Bu sınıfın altında yapılandırıcı dışında bir metoda ihtiyaç yoktur.

2-`public class BagliListeKuyruk` : bu sınıf altında kuyruk bağlı listesinin başlangıç elemanını tutan bir özellik olmalıdır. Bu sınıf altında kuyruğa ekleme, kuyruktan çekme, kuyruk yazdırma, kuyruktaki eleman sayısını döndürme metotları yazılacaktır.

3-`public class AnaSinif` : bu sınıf altında main metodu olacak ve çalışacak ana sınıf olacaktır. Kuyruğa yeni eleman ekleme, kuyruktan silme kuyruk yazdırma ve kuyruktaki eleman sayısını gösteren metotların testleri yapılacaktır.

```
public class Eleman {
      String isim;
      String soyisim;
      int telno;
      int dt;

      Eleman(String isim, String
soyisim, int telno, int dt){
            this.isim=isim;
            this.soyisim=soyisim;
            this.telno=telno;
             this.dt=dt;
      }
}
```