Razredi in objekti
(Classes and objects)

```
public class MyFirstUrn<E> {
    private static final Random RANDOMS = new Random();
    private LinkedList<E> holder;

    public MyFirstUrn () {
        holder = new LinkedList<E> ();
    }
    public void put (E val) {
        holder.add(0, val);
    }
    public E take () {
        int size = holder.size();
        int index = RANDOMS.nextInt(size);
        return holder.remove(index);
    }
    public int size () {
        return holder.size ();
    }
    public boolean contains (E val) {
        return holder.contains (val);
    }
    public void clear () {
        holder.clear();
    }
}
```

Testiranje MyFirstUrn

```
MyFirstUrn<Integer> urn = new MyFirstUrn<Integer> ();

urn.put(2); urn.put(3); urn.put(5); urn.put(7);

System.out.println (urn.take()); System.out.println (urn.take());

urn.put(11); urn.put(13);

System.out.println (urn.take()); System.out.println (urn.take());
```

Možen rezultat:

7
2
13
5

```
private LinkedList<E> holder;

public MyFirstUrn () {
    holder = new LinkedList<E> ();
}
```

Vsak objekt razreda `MyFirstUrn` ima polje `holder`. Vrednost tega polja je objekt knjižničnega razreda `LinkedList`

Polje je privatno. Samo njegov objekt ima dostop do polja.

`MyFirstUrn()` je konstruktor. Ko se prikličemo konstruktorja, ustvarimo nov objekt razreda `MyFirstUrn`.

Konstruktorji se vedno imenujejo z imenom razreda.

`E` je tipski parameter (type parameter)

```
MyFirstUrn<Integer> urn1 = new MyFirstUrn<Integer> ();
MyFirstUrn<Integer> urn2 = new MyFirstUrn<Integer> ();
MyFirstUrn<String> jar = new MyFirstUrn<String> ();
```

`urn1` je nov objekt razreda `MyFirstUrn<Integer>`, ki je boben celnih števil.

`urn2` je še en nov objekt razreda `MyFirstUrn<Integer>` različen od `urn1`. Objekt `urn2` je tudi boben celih števil.

`Integer` je ovojni razred (wrapper class) tipa `int`. Vsak primitiven tip `t` ima svoj ovojni razred, ki je referenčni tip. Objekt ovojnega razreda ima točno eno vrednost. Tip te vrednosti je `t`.

Ovojni razred je potrebno, ker primerek parametra tipa `E` mora biti referenčni tip.

`jar` je nov objekt razreda `MyFirstUrn<String>`, ki je boben nizov.

Ovojni razredi

`boolean`	`Boolean`
`char`	`Character`
`byte`	`Byte`
`short`	`Short`
`int`	`Integer`
`long`	`Long`
`float`	`Float`
`double`	`Double`

Boxing: Če je `n` vrednost tipa `int`, je `new Integer(n)` njegov objekt razreda `Integer`.

Unboxing: Če je `o` objekt razreda `Integer`, je `o.intValue()` ali `(int) o` njegov vrednost tipa `int`.

Po navadi sta 'boxing' in 'unboxing' avtomatično narejeno.

Npr. v `urn.put(2)`, je vrednost `2`, ki ima tip `int`, avtomatično pretvorjena v objekt razreda `Integer` (implicit boxing).

Vsak objekt razreda `MyFirstUrn<E>` ima naslednje metode.

```
public void put (E val)
public E take ()
public int size ()
public boolean contains (E val)
public void clear ()
```

Vse so javne (public) metode. To pomeni, da vsak razred ima dostop do teh metod.

```
void put (E val) { holder.add(0, val); }
E take () {
    int size = holder.size();
    int index = RANDOMS.nextInt(size);
    return holder.remove(index);
}
int size () {return holder.size ();}
boolean contains (E val) {return holder.contains (val);}
void clear () {holder.clear();}
```

Koda uporablja metode knjižničnega razreda `LinkedList<E>`

`void add(int index,E element)`	Inserts the specified element at the specified position in this list.
`int size()`	Returns the number of elements in this list.
`E get(int index)`	Returns the element at the specified position in this list.
`E remove(int index)`	Removes and returns the element at the specified position in the list
`boolean contains(Object o)`	Returns true if this list contains the specified element.
`void clear()`	Removes all of the elements from this list.

The pill jar puzzle

```
final int NUM_PILLS = 20;
final int NUM_TRIALS = 100;

MyFirstUrn<String> jar = new MyFirstUrn<String>();

int total = 0;
for (int trial = 0; trial < NUM_TRIALS; trial++) {
    jar.clear();
    for (int i = 0; i < NUM_PILLS; i++) jar.put("WHOLE");

    while (jar.contains ("WHOLE")) {
        String pill = jar.take();
        if (pill.equals("WHOLE")) jar.put("half");
    }
    int numHalves = jar.size();
    total += numHalves;
    System.out.print ("Trial " + (trial+1) + ": ");
    System.out.println ("number of half pills = " + numHalves);
}
double average = (double)total/NUM_TRIALS;
System.out.println ("Expected number of half pills ~ " + average);
```

```
public class MyFirstStack<E> {
    private LinkedList<E> holder;

    public MyFirstStack () { // posebni konstruktor
        holder = new LinkedList<E> ();
    }
    public void put (E val) {
        holder.add(0, val);
    }
    public E take () { // nova koda
        return holder.remove(0);
    }
    public int size () {
        return holder.size ();
    }
    public boolean contains (E val) {
        return holder.contains (val);
    }
    public void clear () {
        holder.clear();
    }
}
```

```
public class MyFirstQueue<E> {
    private LinkedList<E> holder;

    public MyFirstQueue () {// posebni konstructor
        holder = new LinkedList<E> ();
    }
    public void put (E val) {
        holder.add(0, val);
    }
    public E take () { // nova koda
        int size = holder.size();
        return holder.remove(size-1);
    }
    public int size () {
        return holder.size ();
    }
    public boolean contains (E val) {
        return holder.contains (val);
    }
    public void clear () {
        holder.clear();
    }
}
```

Abstraktni razredi
(Abstract classes)

```
public abstract class LinkedListHolder<E>  {

    protected LinkedList<E> holder;

    public LinkedListHolder () {
        holder = new LinkedList<E> ();
    }
    public void put (E val) {
        holder.add(0, val);
    }

    abstract public E take (); // abstract method

    public int size () {
        return holder.size ();
    }
    public boolean contains (E val) {
        return holder.contains (val);
    }
    public void clear () {
        holder.clear();
    }
}
```

```
public class MyUrn<E> extends LinkedListHolder<E> {
    private static final Random RANDOMS = new Random();
    @Override
    public E take () {
        int size = holder.size();
        int index = RANDOMS.nextInt(size);
        return holder.remove(index);
    }
}

public class MyStack<E> extends LinkedListHolder<E> {
    @Override
    public E take () {return holder.remove(0);}
}

public class MyQueue<E> extends LinkedListHolder<E> {
    @Override
    public E take () {
        int size = holder.size();
        return holder.remove(size-1);
    }
}
```

```
public abstract class LinkedListHolder
```

Abstraktni razred je razred, v katerem je nekaj metod abstraktnih.

```
abstract public E take ();
```

`take` je abstraktna metoda. Abstraktna metoda ima deklaracijo ampak nima kode.

```
protected LinkedList<E> holder;
```

`protected` pomeni, da imajo dostop do polja `holder` samo razredi, ki so podrazredi (subclasses) razreda `LinkedListHolder` (ali v istem paketu (package) razreda `LinkedListHolder`).

`public LinkedListHolder ()`

Ne moremo uporabiti konstruktor abstraktnega razreda. Ta konstruktor bodo podedovali podrazredi razreda `LinkedListHolder`.

`public class MyUrn<E> extends LinkedListHolder<E>`

`MyUrn<E>` je razred, ki je podrazred (subclass) razreda `LinkedListHolder<E>`.

Poleg tega je razred `LinkedListHolder<E>` neposredni nadrazred (immediate superclass) razreda `MyUrn<E>`. To pomeni, da razred `MyUrn<E>` podeduje polja in metode iz razreda `LinkedListHolder<E>`.

Kadar definiramo razred, lahko deklariramo vsaj en razred kot neposredni nadrazred.

```
MyUrn<Integer> urn = new MyUrn<Integer> ();
```

Konstruktor `MyUrn<E>` je podedovan od konstruktorja `LinkedListHolder<E>` iz abstraktnega razreda. (Dedovanje kontruktorjev je možno samo za konstruktorje, ki nimajo parametrov.)

```
@Override
public E take () { ... }
```

Razred `MyUrn<Integer>` implementira abstraktno metodo `take` iz abstraktne razrede `LinkedListHolder<E>`.

`@Override` je pripis. Spomni pogramerja, da ta metoda implementira metodo, ki je deklarirana v (edinem) neposrednem nadrazredu.

Vmesniki
(Interfaces)

```
public interface Holder<E> {

    // puts element into a holder
    public void put (E val);

    // takes one element out of holder
    public E take ();

    // returns number of elements in holder
    public int size ();

    // returns true if the holder contains at least one copy of val
    public boolean contains (E val);

    // empties the holder
    public void clear ();
}

public abstract class LinkedListHolder<E> implements Holder<E> {

    //
    // Same code as before
    //

}
```

- Vmesniki imajo deklaracije metod in polj, ampak metode in polja niso implementirani.
- Razred lahko implementira vmesnik, če ima vse metode in polja, ki so deklarirani v vmesniku.
- Razred lahko implementira mnoge vmesnike.

Type parameter constraints

```
public class MySmallestFirst<E extends Comparable<E>>
        extends LinkedListHolder<E>{

    @Override
    public E take () {
        E smallest = null;
        for (E val: holder) { // for loop using iterator of holder
            if (smallest == null ||
                val.compareTo(smallest) < 0) smallest = val;
        }
        // Remove the first occurrence of smallest in holder
        holder.remove(smallest);
        return smallest;
    }
}
```

`Comparable<E>` je vmesnik za razrede, ki implementirajo metodo `int compareTo(E o)`.