



Midterm



- Utorok 17.4. 18:00, posl. A, čas 90 min.
- prezencia ISIC
- ide o 25 bodov
- test na papieroch,
- 5 príkladov na seprarovných papieroch, opravujeme oddelene
- témy až po prednášku StreamAPI
- vzory midtermov 2012-2017 sú na stránke predmetu
- midterm nie je o syntaxi, ale či veciam rozumiete (kódy nekompilujeme)
- syntax či font zaváži len, ak už nie je možné posúdiť, či rozumiete
- ŽIADNE elektronické pomôcky nie sú dovolené, ani mobil na lavici
- akékoľvek vlastné materiály sú povolené, nesmú „kolovať“ lavicou
- nezničte lesy, čo ste nečítali, vám určite nepomôže
- dozor ochotne odpovie na vaše korektné otázky



Kanonické usporiadanie

(riešenie TamaraS.)

```
public static String[] kanonickeUsporiadane() {
    Comparator<String> zoradPodlaDlzkky =
        (prveSlovo, druheSlovo) -> prveSlovo.length() - druheSlovo.length();
    Comparator<String> zoradPodlaASCII =
        (prveSlovo, druheSlovo) -> prveSlovo.compareTo(druheSlovo);

    List<String> povodnePismena = Arrays.asList(letters);
    return
        IntStream.range(0, povodnePismena.size())
            .filter(ix -> ix >= 'A' && ix <= 'Z')    // indexy písmen A..Z
            .mapToObj(povodnePismena::get)          // ich obraz v Morse
            .sorted(zoradPodlaDlzkky.thenComparing(zoradPodlaASCII))
            .collect(Collectors.toList())
            .toArray(new String[26]);
}
```

Zhoda s vlastným kódom



```
public void add(T start, T end){
    if(start instanceof Integer)
    {
        if(set.isEmpty())
        {
            for(Integer i = (Integer) start; i <= (Integer)end; i++)
            {
                set.add((T) i);
            }
            return;
        }
        if(start.compareTo(set.first()) < 0 && end.compareTo(set.first()) <= 0)
        {
            Integer first = (Integer)start;
            Integer last = (Integer)end;
            for(Integer i=first; i <= last; i++)
            {
                set.add((T)i);
            }
        }
        else if(start.compareTo(set.first()) < 0 && end.compareTo(set.last()) <= 0)
        {
            Integer first = (Integer) set.first();
            for(Integer i=(Integer)start; i <= first; i++)
            {
                set.add((T)i);
            }
        }
        else if(start.compareTo(set.first()) < 0 && end.compareTo(set.last()) > 0)
        {
            Integer first = (Integer)start;
            Integer last = (Integer)end;
            for(Integer i=first; i <= last; i++)
            {
                set.add((T)i);
            }
        }
        else if(start.compareTo(set.first()) > 0 && end.compareTo(set.last()) < 0 && !set.contains(start) || !set.contains(end))
        {
            Integer first = (Integer)start;
            Integer last = (Integer)end;
            for(Integer i=first; i <= last; i++)
            {
                set.add((T)i);
            }
        }
        else if(start.compareTo(set.last()) >= 0)
        {
            for(Integer i=(Integer)start; i <= (Integer)end; i++)
            {
                set.add((T)i);
            }
        }
        if(start.compareTo(set.last()) <= 0 && end.compareTo(set.last()) > 0)
        {
            for(Integer i=(Integer)set.last(); i <= (Integer)end; i++)
            {
                set.add((T)i);
            }
        }
    }
}
// vlozi do množiny interval hodnot [start..end]
```

```
public void remove(T start, T end){
    if(start instanceof Integer)
    {
        if(!set.isEmpty()) {
            if(start.compareTo(set.last()) >= 0)
            {
                return;
            }
            if(end.compareTo(set.first()) <= 0)
            {
                return;
            }
            if(start.compareTo(set.first()) < 0 && end.compareTo(set.last()) <= 0)
            {
                for(Integer i= (Integer)set.first(); i<(Integer)end; i++)
                {
                    set.remove(i);
                }
                return;
            }
            if(start.compareTo(set.first()) > 0 && end.compareTo(set.last()) <= 0)
            {
                for(Integer i= (Integer)start+1; i<(Integer)end; i++)
                {
                    set.remove(i);
                }
                return;
            }
            if(start.compareTo(set.first()) > 0 && end.compareTo(set.last()) > 0)
            {
                for(Integer i= (Integer)start+1; i<(Integer)set.last(); i++)
                {
                    set.remove(i);
                }
                return;
            }
            if(start.compareTo(set.first()) < 0 && end.compareTo(set.last()) > 0)
            {
                set.clear();
                return;
            }
            for(Integer i = (Integer) start+1; i < (Integer)end; i++)
            {
                set.remove((T)i);
            }
        }
    }
}
// zmaze z množiny interval hodnot [start..end]
```

Zhoda s vlastným kódom



```
public void add(T start, T end){
    boolean lock = false;

    //
    //
    System.out.println("ADD start end " + start + " " + end);
    System.out.println(ihm.keySet().isEmpty());
    if (ihm.keySet().isEmpty()) {
        ihm.put(start, end);
    } else {

        for (Entry<T, T> entry : ihm.entrySet()) {
            T key = entry.getKey();
            T value = entry.getValue();
            System.out.println("KEY " + key + " VAL " + value + " SRT " + start + " END " + end);
            System.out.println("compare " + start + " " + key + " " + start.compareTo(key));

            if (start.compareTo(key) < 0) {
                if (end.compareTo(key) < 0) {
                    lock = true; // mensie
                } else if (end.compareTo(key) == 0) {
                    ihm.remove(key, value);
                    ihm.put(start, value);
                    lock = false;
                    break;
                } else if (end.compareTo(key) > 0 && end.compareTo(value) < 0) {
                    ihm.remove(key, value);
                    ihm.put(start, value);
                    lock = false;
                    break;
                } else if (end.compareTo(key) > 0 && end.compareTo(value) == 0) {
                    ihm.remove(key, value);
                    ihm.put(start, end);
                    lock = false;
                    break;
                } else if (end.compareTo(key) > 0 && end.compareTo(value) > 0) {
                    ihm.remove(key, value);
                    ihm.put(start, end);
                    lock = false;
                    break;
                }
            } else if (start.compareTo(key) == 0) {
                if (end.compareTo(key) == 0) {
                    ihm.remove(key, value);
                    ihm.put(key, value);
                    lock = false;
                    break;
                } else if (end.compareTo(key) > 0 && end.compareTo(value) < 0) {
                    ihm.remove(key, value);
                    ihm.put(key, value);
                    lock = false;
                    break;
                } else if (end.compareTo(key) > 0 && end.compareTo(value) == 0) {
                    ihm.remove(key, value);
                    ihm.put(start, end);
                    lock = false;
                    break;
                } else if (end.compareTo(key) > 0 && end.compareTo(value) > 0) {
                    ihm.remove(key, value);
                    ihm.put(key, end);
                    lock = false;
                    break;
                }
            }
        }
    } else if (start.compareTo(key) > 0 && start.compareTo(value) < 0) {
        if (end.compareTo(key) > 0 && end.compareTo(value) < 0) {
            ihm.remove(key, value);
            ihm.put(key, value);
            lock = false;
            break;
        } else if (end.compareTo(key) > 0 && end.compareTo(value) == 0) {
            ihm.remove(key, value);
            ihm.put(start, end);
            lock = false;
            break;
        } else if (end.compareTo(key) > 0 && end.compareTo(value) > 0) {
            ihm.remove(key, value);
            ihm.put(start, end);
            lock = false;
            break;
        }
    }
}
```

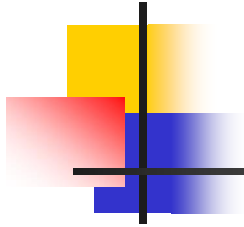
```
public void remove(T start, T end){
    System.out.println("RMV start end " + start + " " + end);

    for (Entry<T, T> entry : ihm.entrySet()) {
        T key = entry.getKey();
        T value = entry.getValue();
        System.out.println("KEY " + key + " VAL " + value + " SRT " + start + " END " + end);
        System.out.println("compare " + start + " " + key + " " + start.compareTo(key));

        if (start.compareTo(key) < 0) {
            if (end.compareTo(key) < 0) {
                lock = true; // mensie
            } else if (end.compareTo(key) == 0) {
                ihm.remove(key, value);
                ihm.put(start, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) < 0) {
                ihm.remove(key, value);
                ihm.put(end, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) == 0) {
                ihm.remove(key, value);
                ihm.put(value, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) > 0) {
                ihm.remove(key, value);
                ihm.put(start, end);
                lock = false;
                break;
            }
        } else if (start.compareTo(key) == 0) {
            if (end.compareTo(key) == 0) {
                ihm.remove(key, value);
                ihm.put(key, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) < 0) {
                ihm.remove(key, value);
                ihm.put(key, key);
                ihm.put(end, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) == 0) {
                ihm.remove(key, value);
                ihm.put(key, key);
                ihm.put(key, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) > 0) {
                ihm.remove(key, value);
                ihm.put(key, key);
                lock = false;
                break;
            }
        } else if (start.compareTo(key) > 0 && start.compareTo(value) < 0) {
            if (end.compareTo(key) > 0 && end.compareTo(value) < 0) {
                ihm.remove(key, value);
                ihm.put(key, start);
                ihm.put(end, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) == 0) {
                ihm.remove(key, value);
                ihm.put(key, start);
                ihm.put(value, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) > 0) {
                ihm.remove(key, value);
                ihm.put(key, start);
                lock = false;
                break;
            }
        } else if (start.compareTo(value) == 0) {
            if (end.compareTo(value) == 0) {
                ihm.remove(key, value);
                ihm.put(start, end);
                lock = false;
                break;
            }
        }
    }
}
```

Refactoring

Extract method





Midterm



- Utorok 17.4. 18:00, posl. A, čas 90-100 min.
- prezencia ISIC
- ide o 25 bodov
- test na papieroch
- 5 príkladov na seprarovných papieroch, opravujeme oddelene
- témy až po prednášku StreamAPI
- vzory midtermov 2012-2017 sú na stránke predmetu
- midterm nie je o syntaxi, ale či veciam rozumiete (kódy nekompilujeme)
- syntax či font zaváži len, ak už nie je možné posúdiť, či rozumiete
- ŽIADNE elektronické pomôcky nie sú dovolené, ani mobil na lavici
- akékoľvek vlastné materiály sú povolené, nesmú „kolovať“ lavicou
- nezničte lesy, čo ste nečítali, vám určite nepomôže
- dozor ochotne odpovie na vaše korektné otázky



Kanonické usporiadanie

(riešenie TamaraS.)

```
public static String[] kanonickeUsporiadane() {
    Comparator<String> zoradPodlaDlzkky =
        (prveSlovo, druheSlovo) -> prveSlovo.length() - druheSlovo.length();
    Comparator<String> zoradPodlaASCII =
        (prveSlovo, druheSlovo) -> prveSlovo.compareTo(druheSlovo);

    List<String> povodnePismena = Arrays.asList(letters);
    return
        IntStream.range(0, povodnePismena.size())
            .filter(ix -> ix >= 'A' && ix <= 'Z')    // indexy písmen A..Z
            .mapToObj(povodnePismena::get)          // ich obraz v Morse
            .sorted(zoradPodlaDlzkky.thenComparing(zoradPodlaASCII))
            .collect(Collectors.toList())
            .toArray(new String[26]);
}
```

Zhoda s vlastným kódom



```
public void add(T start, T end){
    if(start instanceof Integer)
    {
        if(set.isEmpty())
        {
            for(Integer i = (Integer) start; i <= (Integer) end; i++)
            {
                set.add((T) i);
            }
            return;
        }
        if(start.compareTo(set.first()) < 0 && end.compareTo(set.last()) <= 0)
        {
            Integer first = (Integer) start;
            Integer last = (Integer) end;
            for(Integer i=first; i <= last; i++)
            {
                set.add((T) i);
            }
        }
        else if(start.compareTo(set.first()) < 0 && end.compareTo(set.last()) <= 0)
        {
            Integer first = (Integer) set.first();
            for(Integer i=(Integer) start; i <= first; i++)
            {
                set.add((T) i);
            }
        }
        else if(start.compareTo(set.first()) < 0 && end.compareTo(set.last()) > 0)
        {
            Integer first = (Integer) start;
            Integer last = (Integer) end;
            for(Integer i=first; i <= last; i++)
            {
                set.add((T) i);
            }
        }
        else if(start.compareTo(set.first()) > 0 && end.compareTo(set.last()) < 0 && !set.contains(start) || !set.contains(end))
        {
            Integer first = (Integer) start;
            Integer last = (Integer) end;
            for(Integer i=first; i <= last; i++)
            {
                set.add((T) i);
            }
        }
        else if(start.compareTo(set.last()) >= 0)
        {
            for(Integer i=(Integer) start; i <= (Integer) end; i++)
            {
                set.add((T) i);
            }
        }
        if(start.compareTo(set.last()) <= 0 && end.compareTo(set.last()) > 0)
        {
            for(Integer i=(Integer) set.last(); i <= (Integer) end; i++)
            {
                set.add((T) i);
            }
        }
    }
    // vlozi do množiny interval hodnot [start..end]
}
```

```
public void remove(T start, T end){
    if(start instanceof Integer)
    {
        if(!set.isEmpty()) {
            if(start.compareTo(set.last()) >= 0)
            {
                return;
            }
            if(end.compareTo(set.first()) <= 0)
            {
                return;
            }
            if(start.compareTo(set.first()) < 0 && end.compareTo(set.last()) <= 0)
            {
                for(Integer i= (Integer) set.first(); i < (Integer) end; i++)
                {
                    set.remove(i);
                }
                return;
            }
            if(start.compareTo(set.first()) > 0 && end.compareTo(set.last()) <= 0)
            {
                for(Integer i= (Integer) start+1; i < (Integer) end; i++)
                {
                    set.remove(i);
                }
                return;
            }
            if(start.compareTo(set.first()) > 0 && end.compareTo(set.last()) > 0)
            {
                for(Integer i= (Integer) start+1; i < (Integer) set.last(); i++)
                {
                    set.remove(i);
                }
                return;
            }
            if(start.compareTo(set.first()) < 0 && end.compareTo(set.last()) > 0)
            {
                set.clear();
                return;
            }
            for(Integer i = (Integer) start+1; i < (Integer) end; i++)
            {
                set.remove((T) i);
            }
        }
    }
    // zmaze z množiny interval hodnot [start..end]
}
```


Zhoda s vlastným kódom



```
public void add(T start, T end){
    boolean lock = false;
    //
    //
    System.out.println("ADD start end " + start + " " + end);
    System.out.println(ihm.keySet().isEmpty());
    if (ihm.keySet().isEmpty()) {
        ihm.put(start, end);
    } else {
        for (Entry<T,T> entry : ihm.entrySet()) {
            T key = entry.getKey();
            T value = entry.getValue();
            System.out.println("KEY " + key + " VAL " + value + " SRT " + start + " END " + end);
            System.out.println("compare " + start + " " + key + " " + start.compareTo(key));

            if (start.compareTo(key) < 0) {
                if (end.compareTo(key) < 0) {
                    lock = true; // mensie
                } else if (end.compareTo(key) == 0) {
                    ihm.remove(key, value);
                    ihm.put(start, value);
                    lock = false;
                    break;
                } else if (end.compareTo(key) > 0 && end.compareTo(value) < 0) {
                    ihm.remove(key, value);
                    ihm.put(start, value);
                    lock = false;
                    break;
                } else if (end.compareTo(key) > 0 && end.compareTo(value) == 0) {
                    ihm.remove(key, value);
                    ihm.put(start, end);
                    lock = false;
                    break;
                } else if (end.compareTo(key) > 0 && end.compareTo(value) > 0) {
                    ihm.remove(key, value);
                    ihm.put(start, end);
                    lock = false;
                    break;
                }
            } else if (start.compareTo(key) == 0) {
                if (end.compareTo(key) == 0) {
                    ihm.remove(key, value);
                    ihm.put(key, value);
                    lock = false;
                    break;
                } else if (end.compareTo(key) > 0 && end.compareTo(value) < 0) {
                    ihm.remove(key, value);
                    ihm.put(key, value);
                    lock = false;
                    break;
                } else if (end.compareTo(key) > 0 && end.compareTo(value) == 0) {
                    ihm.remove(key, value);
                    ihm.put(start, end);
                    lock = false;
                    break;
                } else if (end.compareTo(key) > 0 && end.compareTo(value) > 0) {
                    ihm.remove(key, value);
                    ihm.put(key, end);
                    lock = false;
                    break;
                }
            }
        }
    } else if (start.compareTo(key) > 0 && start.compareTo(value) < 0) {
        if (end.compareTo(key) > 0 && end.compareTo(value) < 0) {
            ihm.remove(key, value);
            ihm.put(key, value);
            lock = false;
            break;
        } else if (end.compareTo(key) > 0 && end.compareTo(value) == 0) {
            ihm.remove(key, value);
            ihm.put(key, start);
            lock = false;
            break;
        } else if (end.compareTo(key) > 0 && end.compareTo(value) > 0) {
            ihm.remove(key, value);
            ihm.put(start, end);
            lock = false;
            break;
        }
    } else if (start.compareTo(key) > 0 && start.compareTo(value) == 0) {
        ihm.remove(key, value);
        ihm.put(key, value);
        lock = false;
        break;
    } else if (end.compareTo(key) > 0 && end.compareTo(value) == 0) {
        ihm.remove(key, value);
        ihm.put(start, end);
        lock = false;
        break;
    }
}
```

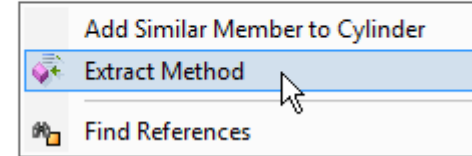
```
public void remove(T start, T end){
    //
    //
    System.out.println("RMV start end " + start + " " + end);
    for (Entry<T,T> entry : ihm.entrySet()) {
        T key = entry.getKey();
        T value = entry.getValue();
        System.out.println("KEY " + key + " VAL " + value + " SRT " + start + " END " + end);
        System.out.println("compare " + start + " " + key + " " + start.compareTo(key));

        if (start.compareTo(key) < 0) {
            if (end.compareTo(key) < 0) {
                lock = true; // mensie
            } else if (end.compareTo(key) == 0) {
                ihm.remove(key, value);
                ihm.put(start, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) < 0) {
                ihm.remove(key, value);
                ihm.put(end, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) == 0) {
                ihm.remove(key, value);
                ihm.put(value, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) > 0) {
                ihm.remove(key, value);
                ihm.put(start, end);
                lock = false;
                break;
            }
        } else if (start.compareTo(key) == 0) {
            if (end.compareTo(key) == 0) {
                ihm.remove(key, value);
                ihm.put(key, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) < 0) {
                ihm.remove(key, value);
                ihm.put(key, key);
                ihm.put(end, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) == 0) {
                ihm.remove(key, value);
                ihm.put(key, key);
                ihm.put(value, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) > 0) {
                ihm.remove(key, value);
                ihm.put(key, key);
                lock = false;
                break;
            }
        } else if (start.compareTo(key) > 0 && start.compareTo(value) < 0) {
            if (end.compareTo(key) > 0 && end.compareTo(value) < 0) {
                ihm.remove(key, value);
                ihm.put(key, start);
                ihm.put(end, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) == 0) {
                ihm.remove(key, value);
                ihm.put(key, start);
                ihm.put(value, value);
                lock = false;
                break;
            } else if (end.compareTo(key) > 0 && end.compareTo(value) > 0) {
                ihm.remove(key, value);
                ihm.put(key, start);
                lock = false;
                break;
            }
        } else if (start.compareTo(value) == 0) {
            if (end.compareTo(value) == 0) {
                ihm.remove(key, value);
                ihm.put(start, end);
                lock = false;
                break;
            }
        }
    }
}
```

Refactoring

Extract method

```
double Cylinder::Volume()  
{  
    return PI * m_radius * m_radius * m_length;  
}
```



Extract method refactoring použijeme, ak:

- ✓ rovnaký, resp. podobný kód sa opakuje v dvoch/viacerých metódach
- ✓ ak je metóda veľmi prepletená s inou triedou, ako v ktorej je definovaná

