Teste POO 13/06/2019

1.

```
public class PolyAsMap implements Poly {
    private int grau;
    private Map<Integer, Double> monomios;

    public PolyAsMap() {
        this.grau = 0;
        this.monomios = new HashMap<>();
    }

    public void addMonomio(int grau, double coeficiente) {
        if (this.grau < grau) this.grau = grau;
        if (this.monomios.containsKey(grau))
            coeficiente += this.monomios.get(grau);
        this.monomios.put(grau, coeficiente);
    }

    public double calcula(double x) {
        return this.monomios.entrySet().stream()
            .mapToDouble((k,v) -> Math.pow(x,k) * v).sum();
    }

    public String toString() {
        StringBuilder sb = new StringBuilder();
        for(int i = grau; i >= 0; i--) {
            if (this.monomios.containsKey(i)) {
                double coef = this.monomios.get(i);
                if (coef > 0) sb.append("+");
                sb.append(coef).append("x^").append(i);
            }
        }
        return sb.toString();
    }
```

2.

a) // Comida

```
public double caloriasGastas(){
    return (this.KmsPercorridos * super.getCaloriasPorUnidadeTempo()) *
        (1 + 0.25 * this.elevacao);
}
// Elíptica
public double caloriasGastas(){
    if (this.esforco <= 4)
        return this.KmsPercorridos * super.getCaloriasPorUnidadeTempo() *
            (1 + 0.2 * this.minutos);
    return this.KmsPercorridos * super.getCaloriasPorUnidadeTempo() *
        (1 + 0.5 * this.minutos);
}
// Abdominais
public double caloriasGastas(){
    return this.numeroRepeticoes * super.getCalorias...();
}
```

b)

```
public double valorTotalCaloriasCliente(String codCliente) throws ClienteNaoExiste{
    if(!this.clientes.containsKey(codCliente)) throw new ClienteNaoExiste();
    return this.clientes.get(codCliente).getMeusExercicios().values()
        .stream().mapToDouble(lista -> lista.stream().
        .mapToDouble(ex -> ex.getAtividade().caloriasGastas()).sum())
        .sum();
}
```

c)

```
public class ClienteNaoExiste extends Exception {
    public ClienteNãoExiste (){
        super();
    }
    public ClienteNaoExiste (String mensagem){
        super (mensagem);
    }
}
...
public double totalKms (Cliente (String cod) Cliente, LocalDate dataExercicio)
    thaows ClienteNaoExiste, ExercicioInexistente {
    if (! this.clientes.containsKey (cod) (Cliente)) throw new (ClienteNãoExiste();
    Cliente c = this.clientes.get (cod) (Cliente);
    if (! c.getMeusExercicios(). containsKey (dataExercicio))
        throw new ExercicioInexistente ();
    return c.getMeusExercicios(). get (dataExercicio). stream()
        .map (Exercicio :: getAtividade)
        .filter (a -> a instanceof ComDistancia)
        .mapToDouble (a -> a.getKmsPercorridos()). sum();
}
```

d)

```
public boolean existeProfessor (String prof){
    for (Cliente c : this.clientes.values())
        for (List<Exercicio> ls : this.getMeusExercicios(). values())
            if (ls.stream().anyMatch (ex -> ex.getProfessor()
                .equals(prof))) return true;
    return false;
}
```

3.

```
public abstract class Actividade implements Comparable<Actividade>, Serializable {
    ...
    public int compareTo(Actividade a){
        return this.caloriasGastas() - a.caloriasGastas();
    }
}
```

4.

```
public Map<String, List<Exercicios>> exerciciosPorProf(){
    Map<String, List<Exercicios>> mapa = new HashMap<>();
    this.meusExercicios.values().forEach(ex -> {
        if (!mapa.containsKey(ex.getProfessor()))
            mapa.put(ex.getProfessor(), new ArrayList<>());
        mapa.get(ex.getProfessor()).add(ex);
    });
    return mapa;
}
```

5.

```
public String professorMaisExistente(){
    Map<String, Integer> mapa = new HashMap<>();
    for(Cliente c :: this.clientes.values()){
        for(List<Exercicio> ls :: c.getMeusExercicios().values()){
            for(Exercicio ex :: ls){
                if (!mapa.containsKey(ex.getProfessor()))
                    mapa.put(ex.getProfessor(), 0);
                mapa.put(ex.getProfessor(), mapa.get(ex.getProfessor()),
                    + ex.caloriasGastas());
            }
        }
    }
    return mapa.keySet().stream().max((a,b) -> mapa.get(b) - mapa.get(a)).get();
```

6.

```
public class GrowingArrayActividade implements Serializable, Comparable {
    private Actividade[] lista;
    private int tamanho;

    public Actividade get (int indice) throws IndiceErrado {
        if ( this.tamanho >= indice) .
            throw new IndiceErrado();
        return lista[indice].clone();
    }

    public void add (Actividade a){
        Actividade[] novaLista = new Actividade[++(this.tamanho)];
        System.arraycopy(this.lista, 0, novaLista, 0, this.tamanho -1);
        novaLista[this.tamanho -1] = a.clone();
        this.lista = novaLista;
    }

    public static GrowingArrayOfActividade leGra...dada (String fich)
        throws IOException {
        FileOutputStream fos = new FileOutputStream(fos);
        ObjectOutputStream oos= new ObjectOutputStream (fos);
        oos.writeObject(this);
        oos.close();
        fos.close();
    }
```