UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica

etsinf

# Pràctica 5

# Selecció: mètode `cross` de la classe `Point`

Introducció a la Informàtica i a la Programació (IIP)

Curs 2019/20

Departament de Sistemes Informàtics i Computació

Marisa Llorens

mllorens@dsic.upv.es

# 5 Activitats de laboratori

## Activitat 1: crear el paquet BlueJ `pract5`

1. Descarrega al directori `Downloads` els fitxers `Point.java` i `RayTest.java`, disponibles a la carpeta de material per a la pràctica 5 de *PoliformaT*.

2. Obre el projecte *BlueJ* de treball de l'assignatura (`iip`).

3. Crea un nou paquet (*Edició - Nou Paquet*) de nom `pract5` i obre'l (doble clic).

4. Agrega al paquet `pract5` les classes `Point` i `RayTest` (*Edició - Afegir Classe des d'Arxiu*). Comprova que les seues primeres línies inclouen la directiva `package pract5;`, que indica que són classes del paquet.

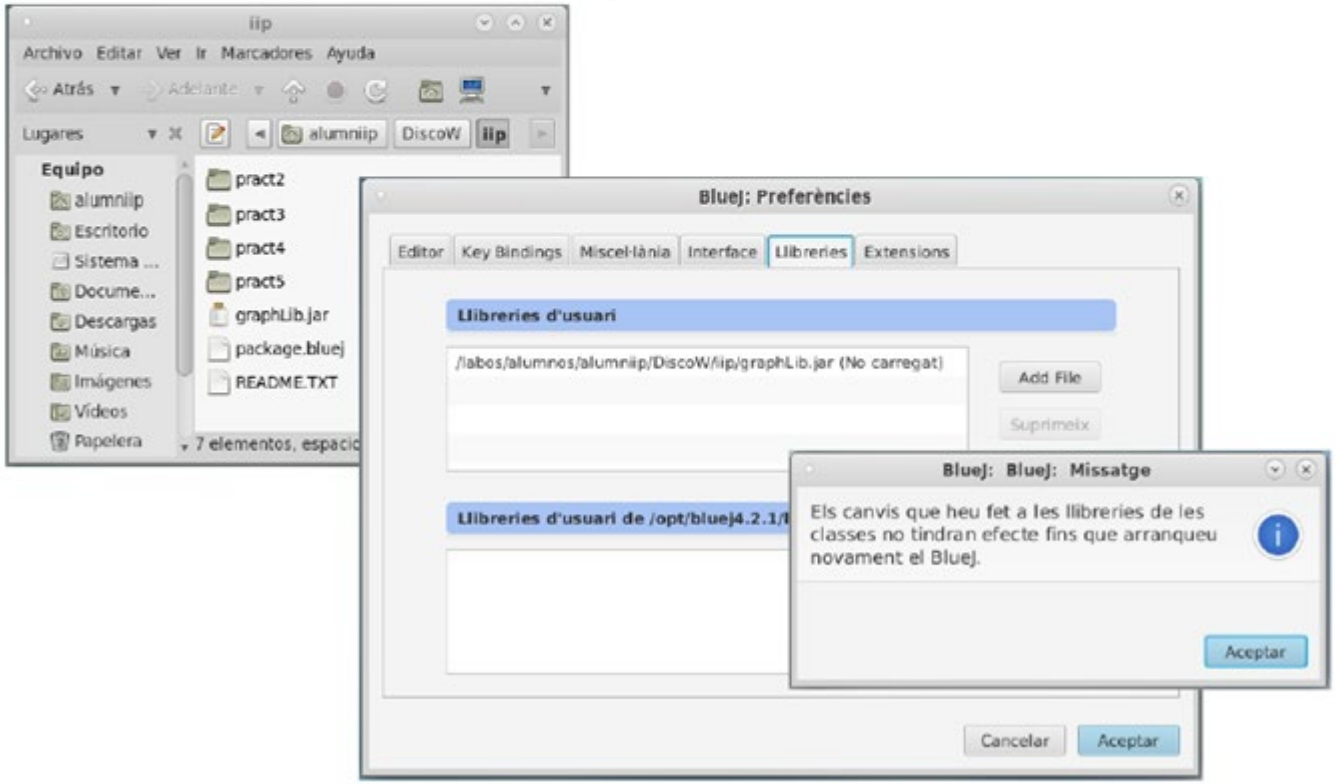
## Activitat 2: instal·lació de la llibreria gràfica `Graph2D`

Per poder mostrar gràficament els segments i els encreuaments dels raigs, se't proporciona una llibreria gràfica que permet representar gràficament punts i línies, entre altres elements, en un espai bidimensional. Es tracta d'una llibreria desenvolupada a propòsit, en l'àmbit de les assignatures IIP i PRG, per facilitar als alumnes de primer curs l'obtenció de resultats gràfics de forma senzilla.

La llibreria gràfica (classe `Graph2D` del paquet `graph2D`) es facilita com una llibreria en el fitxer `graphLib.jar` (disponible a la carpeta *IIP:recursos/Laboratorio/Librería gráfica* de *PoliformaT*). Has de carregar aquesta llibreria com segueix:

1. Situa el fitxer (`graphLib.jar`) al projecte de pràctiques (`iip`).

2. En *Preferències-Llibreries* de *BlueJ*, afegeix el fitxer `graphLib.jar`. Hauràs d'engegar novament *BlueJ* perquè la llibreria es carregue, com pots observar a la Figura 4.




La documentació de la classe `Graph2D` se't proporciona al fitxer `docGraph2D.zip`. Descomprimeix-lo en el projecte `iip` per tal de consultar aquesta documentació (fitxer `Graph2D.html`) quan siga necessari.

- PROBLEMA: Com comprovar si un punt és interior a un polígon?

Algorisme del raig: avançar des del punt en una direcció fixa, per exemple, paral.lela a l'eix X en sentit positiu, i comptar el nombre de vegades que es creua algun costat del polígon. Si aquest nombre és parell, el punt és exterior i si és senar, el punt és interior.
Formalment, es tracta d'una aplicació del *Teorema de Jordan*.

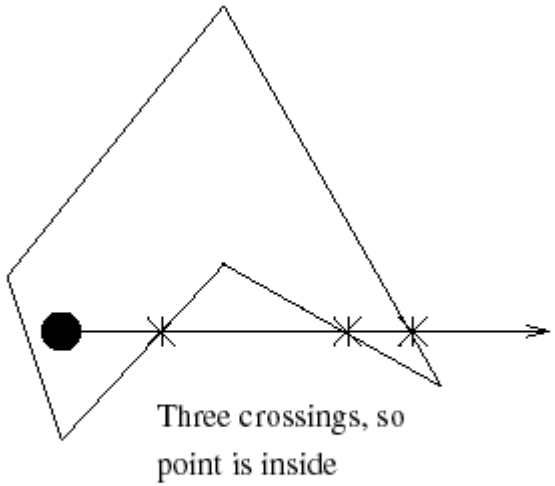
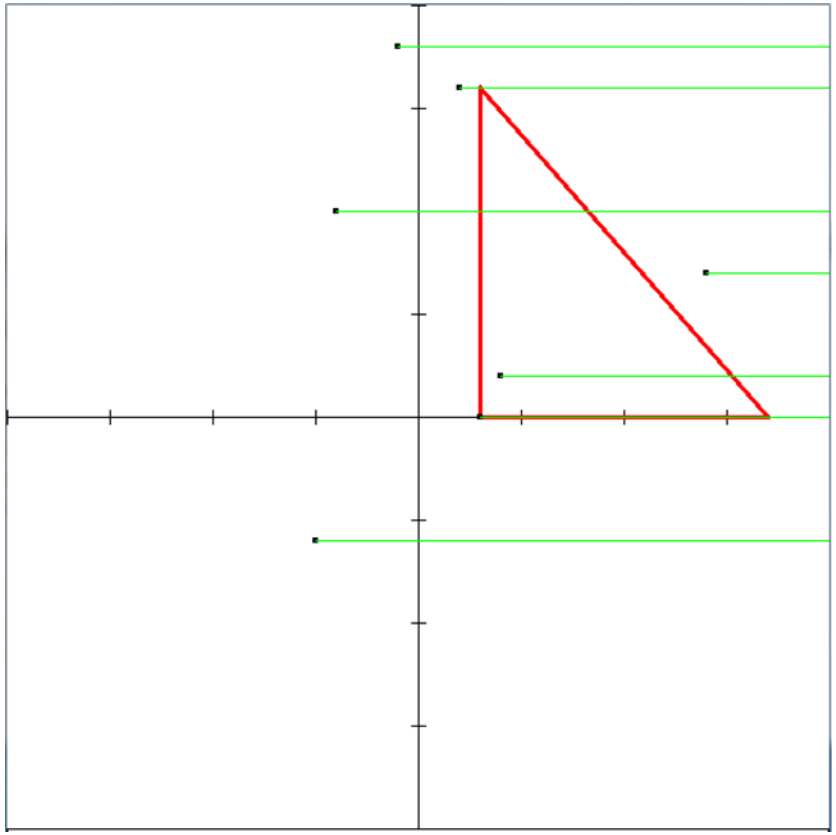



Figure 1 - Crossings Test

http://erich.realtimerendering.com/ptinpoly/

- PROBLEMA DE LA PRÀCTICA 5: implementar un mètode que, donat un punt **p**, comprove si un raig iniciat a **p** creua un segment de recta delimitat pels punts **u** i **v**.

- Si es té un polígon donat per una seqüència de vèrtexs, el mètode permet comprovar si **p** és interior al polígon pel mètode del raig.
- S'ha de comptar el nombre de costats del polígon travessats pel raig i comprovar si és parell o senar (anant amb compte si el raig travessa el polígon per un vèrtex, ja que el vèrtex pertany a dos costats diferents).
- L'algorisme per comptar el nombre d'encreuaments s'implementarà en la pràctica 7.



# 3 Disseny de les classes de l'aplicació

Per a la resolució del problema plantejat, es completarà la implementació de les següents classes:

- La classe *"Tipus de Dades"* `Point`, que representa un punt al pla cartesià mitjançant els següents atributs i mètodes:
  - Atributs públics: les constants Java –variables finals (`final`) de classe (`static`)– de tipus `int` que defineixen l'encreuament: `DONT_CROSS`, `LOW_CROSS`, `CROSS`, `HIGH_CROSS`, amb valors `-1`, `0`, `1` i `2`, respectivament.
  - Atributs privats: les variables d'instància `x` i `y` de tipus `double` que defineixen l'abscissa i l'ordenada del punt, respectivament.
  - Mètodes públics: constructor, *getters*, *setters*, `toString`, `equals` i `cross`.

    El mètode `cross`, donats dos objectes de tipus `Point`, `u` i `v`, que representen els punts extrems d'un segment de recta $\overrightarrow{uv}$, comprova si el raig, que s'inicia en `this` i avança paral·lel a l'eix $X$ en sentit positiu, creua el segment de recta $\overrightarrow{uv}$, és a dir, passa per un únic punt del segment. A la secció 4 es descriu amb detall l'anàlisi de casos necessària per la seua implementació.
- La classe *"Programa"* `RayTest`, en el mètode `main` de la qual es prova el mètode `cross` de la classe `Point` i es mostra el resultat que retorna en l'eixida estàndard i en l'eixida gràfica.

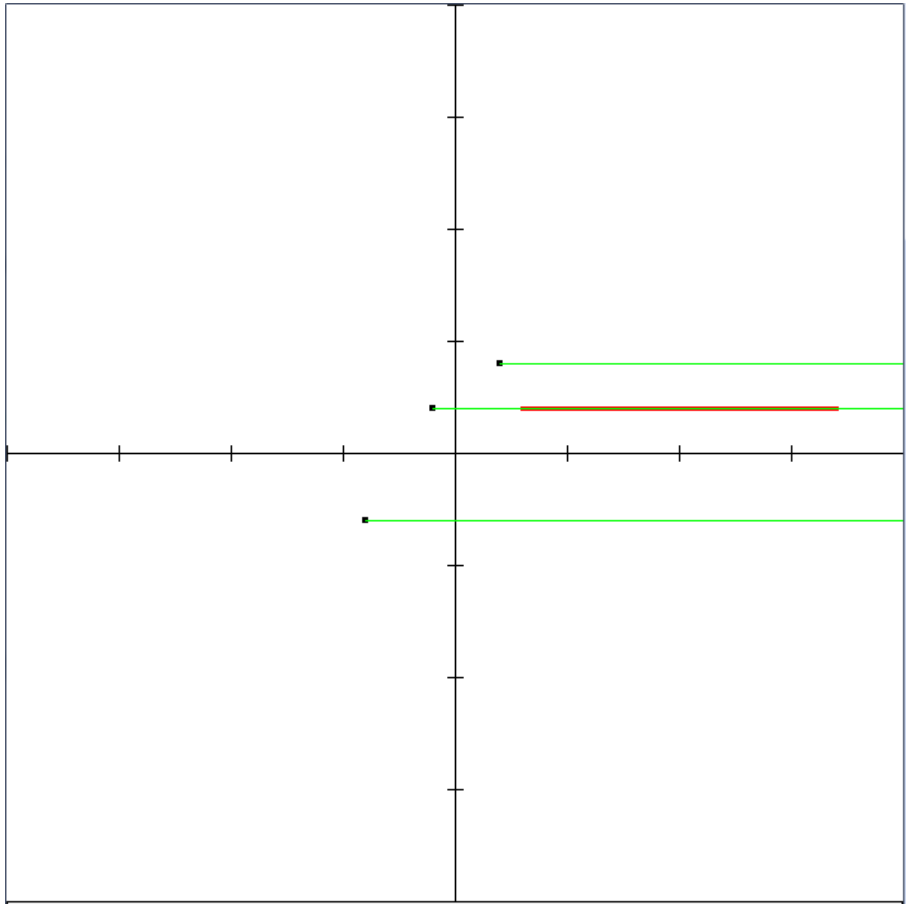
# 4 Disseny del mètode cross

El perfil del mètode **cross** és el següent:

```
/** Donat el raig que s'inicia en this i avança paral.lel a l'eix X
 *  en sentit +, comprova si aquest raig creua el segment de
 *  recta uv, es a dir, passa per un unic punt del segment.
 *  @param u Point, punt extrem del segment de recta uv.
 *  @param v Point, punt extrem del segment de recta uv.
 *  @return int, enter entre DONT_CROSS (-1), LOW_CROSS (0),
 *  CROSS (1), HIGH_CROSS (2), segons els casos:
 *  - Si el raig no creua el segment, torna DONT_CROSS.
 *  - Si el raig el creua per l'extrem més baix, torna LOW_CROSS.
 *  - Si el raig el creua per un punt entre u i v, torna CROSS.
 *  - Si el raig el creua per l'extrem més alt, torna HIGH_CROSS.
 */
public int cross(Point u, Point v)
```

Aquest mètode es pot abordar calculant per on travessa el raig la recta definida per `u` i per `v`, tenint en compte el pendent d'aquesta recta. Suposem, en primer lloc, que s'ha seleccionat en una variable `pHigh` el punt d'entre `u` i `v` més alt (el de major ordenada), i en una variable `pLow` el més baix (el de menor ordenada):

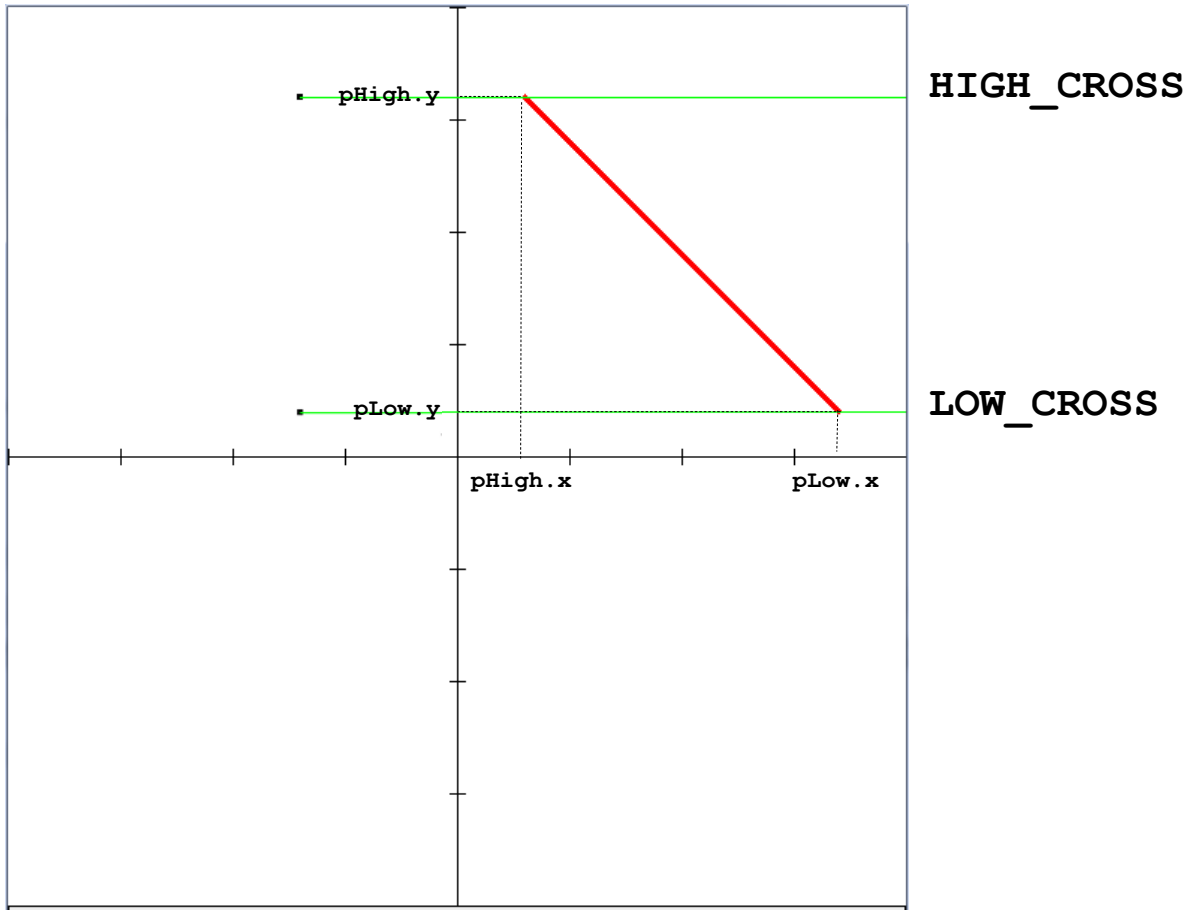
- **CAS 1**: raig i segment són || (**`pHigh.y == pLow.y`**), aleshores, cap raig creua el segment, ja que no passa per cap punt del segment, o passa per tots



Si el raig no és || al segment:

**CAS 2**: el raig passa per **`pHigh`** **`(this.y == pHigh.y && this.x <= pHigh.x)`**

**CAS 3**: el raig passa per **`pLow`** **`(this.y == pLow.y && this.x <= pLow.x)`**



Si el raig no passa ni per **`pHigh`** ni per **`pLow`**:

calcular el punt de tall del raig amb la recta que conté el segment. Siga aquesta recta $ax + b$, i (`xCut`, `yCut`) el punt de tall que es desitja calcular. Tenint en compte que

$$\texttt{yCut} = \texttt{this.y},$$

$$a = \frac{\texttt{pHigh.y} - \texttt{pLow.y}}{\texttt{pHigh.x} - \texttt{pLow.x}},$$

$$b = \texttt{pLow.y} - a \cdot \texttt{pLow.x},$$

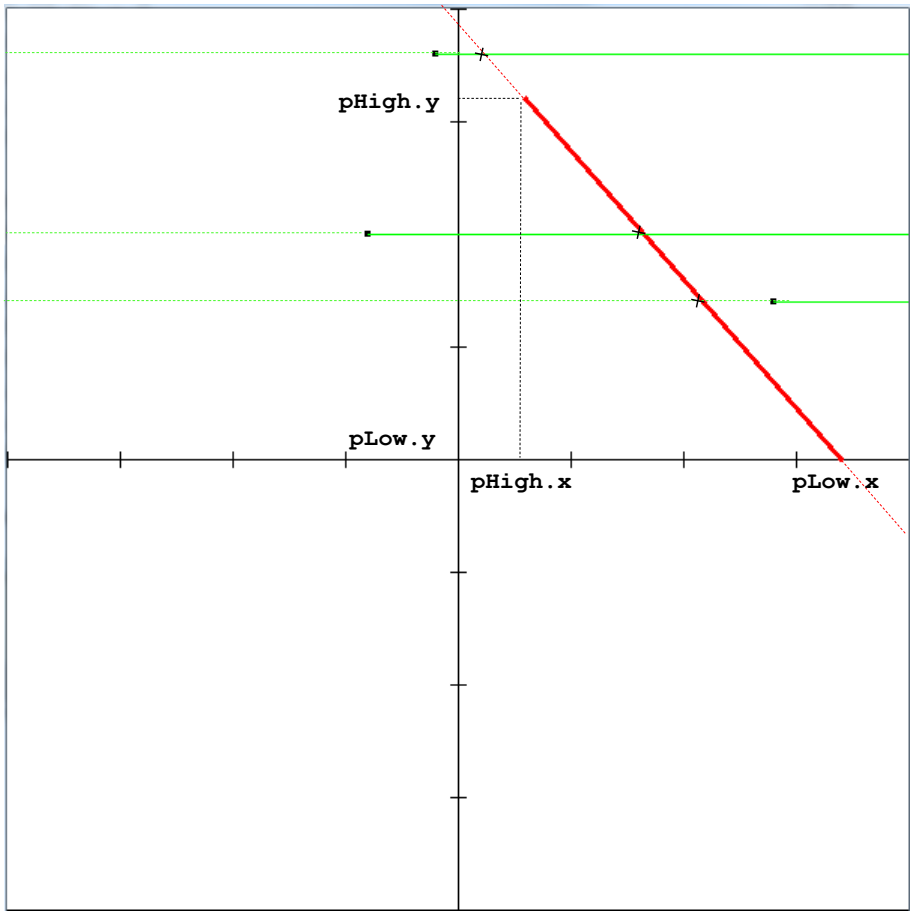
$$\texttt{xCut} = \frac{\texttt{yCut} - b}{a}$$

aleshores, l'abscissa del punt de tall es calcula com:

```
double xCut = (this.y - pLow.y) * (pHigh.x - pLow.x)
            / (pHigh.y - pLow.y) + pLow.x;
```

**CAS 4**: el raig creua el segment perquè **`yCut`** es troba entre l'ordenada més alta i la més baixa del segment, i **`xCut`** es troba a la dreta de **`this.x`**
(**`this.x <= xCut && pLow.y < yCut && yCut < pHigh.y`**)

**CAS 5**: el raig NO creua el segment perquè NO es compleix l'anterior
(**`!(this.x <= xCut && pLow.y < yCut && yCut < pHigh.y)`**)

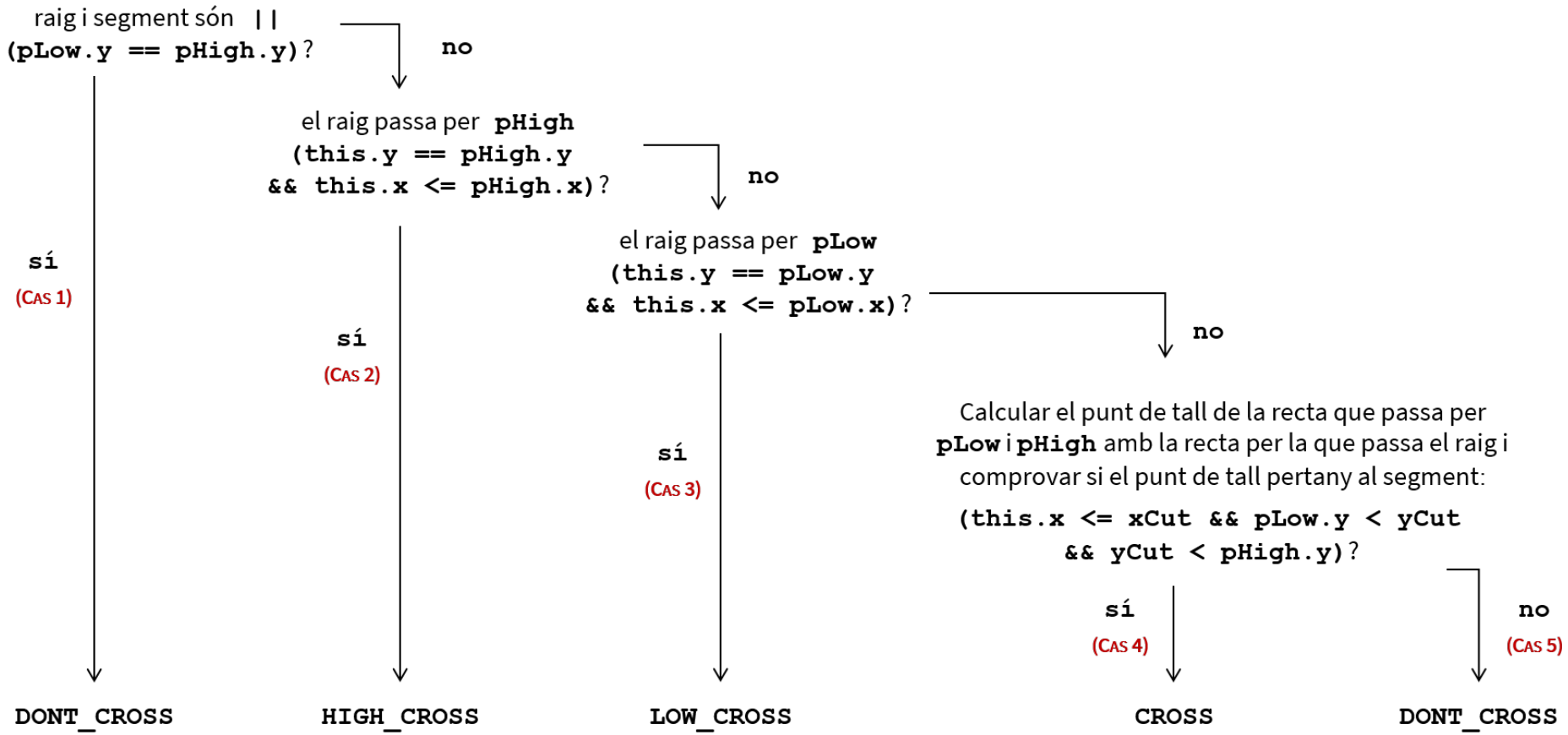



**`DONT_CROSS`**

**`CROSS`**

**`DONT_CROSS`**

# Anàlisi de casos del mètode `cross`:



## Activitat 3: completar la classe `Point`

Al paquet `pract5`, completa la classe `Point` seguint al peu de la lletra els comentaris que apareixen en el seu cos. En concloure, la classe ha de contenir:

1. Les variables d'instància i de classe que s'indiquen en la secció 3.
2. Un mètode constructor per defecte que crea el punt (0.0, 0.0) i un mètode constructor que crea un punt de coordenades donades com a paràmetres.
3. Els mètodes `get` i `set` associats a cadascuna de les variables d'instància.
4. El mètode `distance` que torna la distància entre el punt `this` i un altre punt donat.
5. El mètode `move` que actualitza les coordenades del punt `this` als valors donats com paràmetres.
6. El mètode `equals` que comprova si dos objectes de tipus `Point` són iguals, és a dir, si tots els seus atributs coincideixen.
7. El mètode `toString` que torna un `String` representant el punt `this` en el format típic matemàtic, i.e., `(x,y)`.
8. El mètode `cross` que implementa l'esquema de anàlisi de la Figura 3 per tal de calcular l'encreuament.

```
public int cross(Point u, Point v) {
    int cut;
    Point pLow, pHigh;
    /* COMPLETAR: inicialitzar pLow i pHigh */

    /* COMPLETAR: assignar un valor a cut seguint l'analisi
     * de casos del butlleti */

    return cut;
}
```

## Activitat 4: completar la classe `RayTest` - eixida estàndard

La classe `RayTest` és la classe *"Programa"* en la que es comprova si el mètode `cross` de la classe `Point` és correcte. Per a això, s'escriuran els mètodes següents:

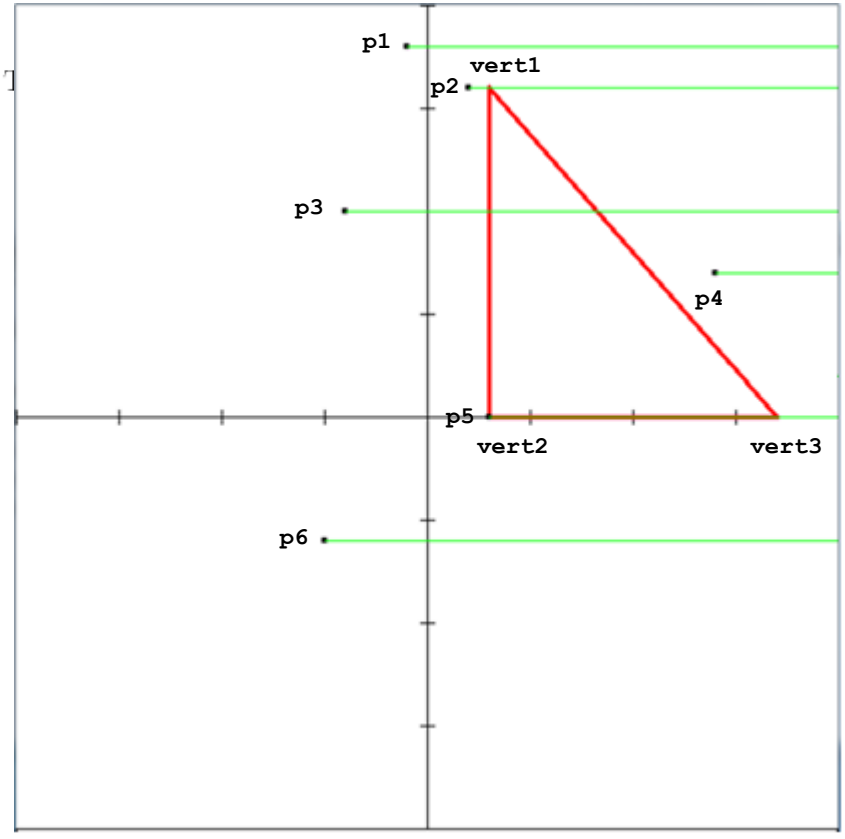
- Mètode `main`, on es declaren tres vèrtexs:

```
Point vert1 = new Point(3.0, 16.0),
    vert2 = new Point(3.0, 0.0),
    vert3 = new Point(17.0, 0.0);
```

que determinen els costats del triangle de la Figura 1, de manera que el segment delimitat per `vert2` i `vert3` és paral·lel als raigs i permet provar el **Cas 1** de l'anàlisi de casos, mentre que es pot fer servir qualsevol dels altres dos segments per provar la resta de casos. Es declaren a més uns punts a testejar:

```
Point p1 = new Point(-1.0, 18.0), p2 = new Point(2.0, 16.0),
    p3 = new Point(-4.0, 10.0), p4 = new Point(14.0, 7.0),
    p5 = new Point(1.0, 0.0), p6 = new Point(-5.0, -6.0);
```

amb els quals es poden cobrir tots els possibles casos del mètode, com a la T




Taula 1: Casos de prova per a `p.cross(u, v)`.

| Cas | Exemples | u | v | p |
|---|---|---|---|---|
| 1 | a | vert2 | vert3 | qualsevol |
| 2 | a | vert1 | vert2 | p2 |
| | b | vert1 | vert3 | p2 |
| 3 | a | vert1 | vert2 | p5 |
| | b | vert1 | vert3 | p5 |
| 4 | a | vert1 | vert2 | p3 |
| | b | vert1 | vert3 | p3 |
| 5 | a | vert1 | vert2 | p1 |
| | b | vert1 | vert2 | p4 |
| | c | vert1 | vert2 | p6 |
| | d | vert1 | vert3 | p1 |
| | e | vert1 | vert3 | p4 |
| | f | vert1 | vert3 | p6 |

En el mètode es realitzaran, almenys, cinc crides, escollides de manera que es cobreixin els cinc casos possibles de l'algorisme (en el codi se't dóna resolt, com a guia, el **Cas 1**). Per a cada crida, es mostrarà el resultat a l'eixida estàndard en un format com el del següent exemple, que ha pres els casos de prova **1a**, **2b**, **3b**, **4b** i **5e** de la Taula 1:

```
Encreuament del segment (3.0,0.0) a (17.0,0.0) des de (-1.0,18.0) : DONT_CROSS
Encreuament del segment (3.0,16.0) a (17.0,0.0) des de (2.0,16.0) : HIGH_CROSS
Encreuament del segment (3.0,16.0) a (17.0,0.0) des de (1.0,0.0) : LOW_CROSS
Encreuament del segment (3.0,16.0) a (17.0,0.0) des de (-4.0,10.0) : CROSS
Encreuament del segment (3.0,16.0) a (17.0,0.0) des de (14.0,7.0) : DONT_CROSS
```

Nota que, en aquesta eixida, el resultat del mètode `cross` es mostra amb la paraula corresponent `DONT_CROSS`, `HIGH_CROSS`, `LOW_CROSS`, `CROSS` en lloc del valor enter que torna el mètode. Amb aquest propòsit, es farà servir el mètode auxiliar `crossToString`, el codi del qual s'ha de completar tal com es descriu a continuació.

- Mètode `crossToString` on, **fent ús d'una instrucció `switch` i de les constants definides a la classe `Point`**, has d'actualitzar adequadament el valor de la variable `res`.

Una vegada completada i provada aquesta classe, pots millorar el seu codi implementant un mètode estàtic `showCross` que, donats tres `Point` que representen un punt i els extrems d'un segment, incloga la crida al mètode `cross` i l'eixida per pantalla que es realitzen per a cadascun dels casos provats.

## Activitat 5: completar la classe `RayTest` - eixida gràfica

Fent ús de la llibreria `Graph2D` del paquet `graph2D`, es van a mostrar els casos de prova en una eixida gràfica, de manera que es puguen contrastar amb els resultats obtinguts en l'activitat anterior. Nota que, al començament de la classe `RayTest`, s'ha inclòs la directiva d'importació de la classe gràfica:

```
import graph2D.Graph2D;
```

Una vegada fet això, ja és possible definir objectes d'aquesta classe i operar sobre ells a la classe `RayTest`. Com es veu al codi que se't proporciona, per crear l'espai de dibuix, s'usa el constructor que crea un `Graph2D` a partir de les dimensions reals dels eixos de coordenades (`-20, 20` per a l'eix $X$ i `-20, 20` per a l'eix $Y$), les dimensions de la finestra (`600` $\times$ `600`), un color de fons (`Color.WHITE`) i un títol.

A continuació, el mètode d'instància `drawLine`, a partir de les coordenades dels punts `u` i `v`, un color (`Color.RED`) i un grossor (`3`), dibuixa el segment $\overrightarrow{uv}$ a l'espai de dibuix.

Per últim, s'invoca al mètode privat estàtic `drawRay` (de la classe `RayTest`) per tal de dibuixar cada un dels punts a testejar (amb el mètode `drawPoint`) i els seus raigs (amb el mètode `drawLine`). El codi d'aquest mètode auxiliar `drawRay` és el que segueix:

```
/** Dibuixa a l'espai de dibuix gd el Point p i el seu raig. */
private static void drawRay(Graph2D gd, Point p) {
    gd.drawPoint(p.getX(), p.getY(), Color.BLACK, 4);
    gd.drawLine(p.getX(), p.getY(), 20, p.getY(), Color.GREEN, 1);
}
```

En aquesta activitat has de completar el dibuix dels raigs de prova, per a visualitzar el seu creuament amb el segment $\overrightarrow{uv}$. Igual que per a l'eixida estàndard, al codi se't proporciona, com a guia, la resolució del **Cas 1**.

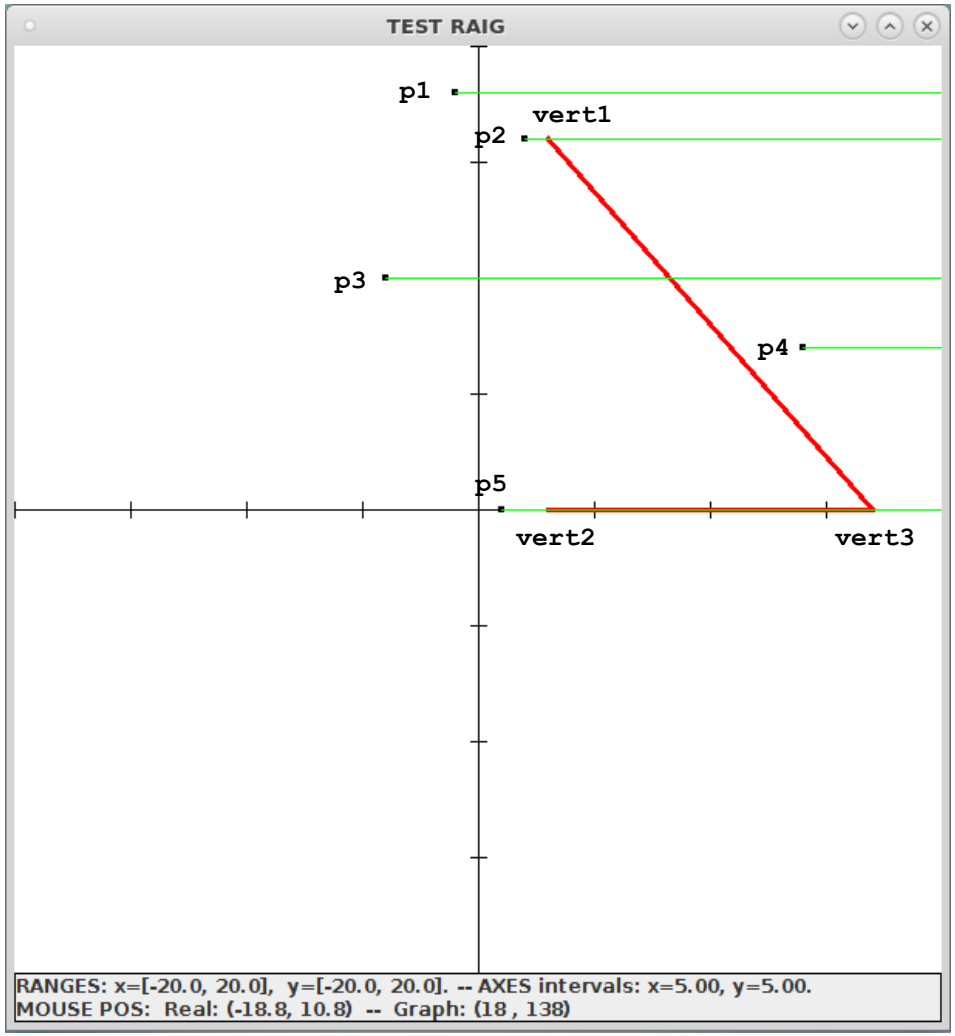
El resultat en l'eixida gràfica del mètode `main`, per exemple, per als casos de prova **1a**, **2b**, **3b**, **4b** i **5e** de la Taula 1, ha de ser el que es mostra a la Figura 5(b).

BlueJ: Terminal Window - iip

Options

```
Encreuament del segment (3.0,0.0) a (17.0,0.0) des de (-1.0,18.0) : DONT_CROSS
Encreuament del segment (3.0,16.0) a (17.0,0.0) des de (2.0,16.0) : HIGH_CROSS
Encreuament del segment (3.0,16.0) a (17.0,0.0) des de (1.0,0.0) : LOW_CROSS
Encreuament del segment (3.0,16.0) a (17.0,0.0) des de (-4.0,10.0) : CROSS
Encreuament del segment (3.0,16.0) a (17.0,0.0) des de (14.0,7.0) : DONT_CROSS
```

Can only enter input while your programming is running




| | | | | | |
|---|---|---|---|---|---|
| DONT_CROSS | 1 | a | vert2 | vert3 | p1 |
| HIGH_CROSS | 2 | b | vert1 | vert3 | p2 |
| CROSS | 4 | b | vert1 | vert3 | p3 |
| DONT_CROSS | 5 | e | vert1 | vert3 | p4 |
| LOW_CROSS | 3 | b | vert1 | vert3 | p5 |

## Activitat 6: comprovar l'estil de les classes `Point` i `RayTest`

Comprova que el codi de les classes escrites compleix les normes d'estil usant el `Checkstyle` de *BlueJ*, i corregeix-lo si no és el cas.

## Activitat 7: validar la classe `Point`

Quan el teu professor ho considere convenient, deixarà disponible en *PoliformaT* una classe de prova (o *Unit Test*) per a validar el codi de la teua classe `Point`. En general, per a passar els tests correctament, cal assegurar-se que s'usen els mateixos identificadors d'atributs i mètodes proposats en aquest document, seguint estrictament les característiques sobre modificadors i paràmetres proposats en la seua capçalera. Segueix aquests passos:

1. Descarrega l'arxiu `PointUnitTest.class` sobre el directori del paquet `pract5` i reobre el teu projecte `iip` des de *BlueJ*.
2. Tria l'opció *Test All* del menú contextual que apareix en fer clic amb el botó dret del ratolí sobre la icona de la classe *Unit Test*. S'executaran un conjunt de proves sobre els mètodes implementats de la classe `Point`, comparant resultats esperats amb els realment obtinguts.
3. Si els mètodes estan bé, apareixeran marcats amb el símbol ✓ (green color) en la finestra *Test Results* de *BlueJ*. Per contra, si algun dels mètodes no funciona correctament, llavors apareixerà marcat mitjançant el símbol `X`. Si selecciones qualsevol de les línies marcades amb `X`, en la part inferior de la finestra es mostra un missatge més descriptiu sobre la possible causa d'error.
4. Si, després de corregir errors i recompilar la classe, la icona de la *Unit Test* està ratllada a quadres, llavors tanca i torna a obrir el projecte *BlueJ*.