Aplicacions Interactives Multiplataforma

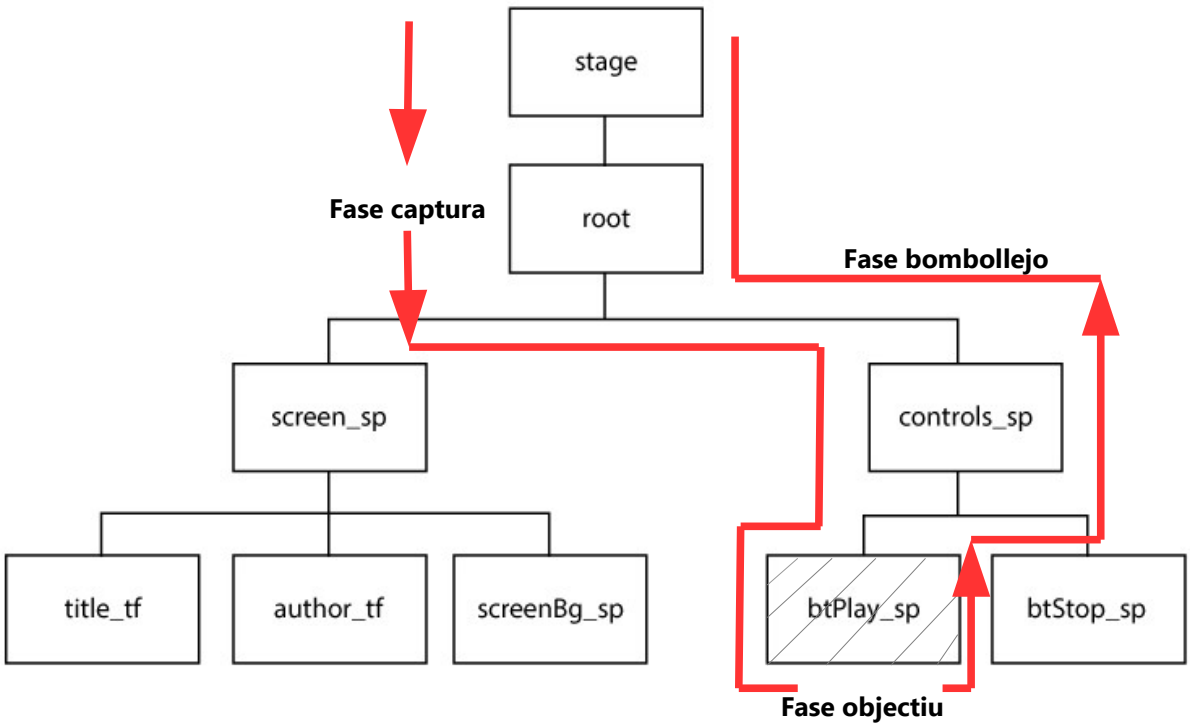
Prova d'avaluació continuada #2

Nom: Aitor Javier
Cognoms: Santaeugenia Marí
Data lliurament: 04/12/2016

PRIMERA PART

Pregunta 1.

Quan parlem d'esdeveniments en *displayObjects* és important conèixer el seu *Event Flow* (recorregut de l'esdeveniment).
Donada la *displayList lectorAudioDisplayList.jpg* per un simple lector d'àudio, descriu el *event flow* si fem clic a l'objecte *btPlay_sp*
Cita un avantatge que ens ofereix el *event flow* per a la realització d'aquesta aplicació.



- *Flash player* o *AIR* distribueix els objectes quan algun *event* ocorre, en ell, *l'event flow* descriu com un objecte es mou a través del *display list*, la qual està ordenada en una jerarquia que pot ser descrita com a arbre.
- Quan *Flash Player* o *Air* distribueix un *event* d'un objecte per a un *event* relacionat a la *display list*, aquest objecte realitza un viatge d'anada i tornada des de el *stage* fins al node objectiu. Així doncs, al clicar sobre *btPLay_SP* és distribuirà un *event* objecte "*btPlay_sp"* com a node objectiu.
- El flux d'esdeveniments està normalment dividit en tres parts: fase captura que compren la etapa inicial fins al node objectiu; fase objectiu que consisteix únicament amb el node objectiu; i fase de bombollejo que compren els nodes trobats en el viatge de retorn des de el pare del node objectiu fins al *stage*.
- Gràcies al ***event flow*** és permet un control de esdeveniments més potent que altres llenguatges de programació, doncs podem agregar *listeners* d'esdeveniments no només a un node destí o objectiu, sinó també a qualsevol node al llarg del flux d'esdeveniments. En el nostre cas, per exemple, pot ser útil per afegir un focus al *placeholder* del botó o al text del botó per tal de poder-ho canviar a gust en el mateix esdeveniment.

Pregunta 2.

Si volem detectar el canvi de dimensions d'una ARM d'escriptori en AIR.
Quin esdeveniment haurem escoltar?
Qui ha d'escoltar aquest esdeveniment?
Si volem utilitzar aquesta mateixa *ARM* en dispositius mòbils, com podríem aprofitar aquest mateix esdeveniment per detectar en quina orientació (*portrait* o *landscape*) es troba el dispositiu?

- Aquest exercici ja l'hem realitzat per exemple en la pràctica#1. Per això, l'esdeveniment a escoltar per tal de canviar les mides o dimensions d'una ARM hauria de ser ***Event.resize***.
- Aquest esdeveniment l'hauria d'escoltar el *stage*, normalment dins del *init* de la classe principal o de les classes de les diferents pantalles que estiguin emprant.
- Per tal de saber la orientació, tant si es *potrait* o *landscape* només caldria que en la cridada de la funció del esdeveniment *Event.resize*, tinguéssim unes variables *width* o *height* (amplada i alçada) i que diguéssim que si l'amplada es major que l'alçada, llavors tocaria reordenar tots els elements de l'aplicació.

Pregunta 3.

Quan descarreguem imatges dins d'una ARM és important informar a l'usuari del que està passant, anar informant del començament de la descarrega, del percentatge de la imatge ja descarregada i de la finalització de la descarrega.
Indica quines classes i esdeveniments ens permetran realitzar aquesta *monitorització* de la descàrrega.
Explica els passos que seguiríem.

- Aquest exercici l'hem realitzat en la part pràctica. Així doncs,
- Primer de tot, importar el paquet

```
import flash.display.Loader;
```

- Tot seguit, crear una variable del objecte *Loader*

```
// Loader nos permite cargar una imagen
var loader:Loader = new Loader();
```

- Crear les passes per la càrrega de la imatge, com crear la variable *string*, inicialitzar la imatge a carregar i crear l'objecte *URLRequest* de la variable emprada per la imatge.

```
var url:String;

url = _galleryData[0].imageUrl;

// Creamos el urlRequest para utlizar con loader
var urlImage:URLRequest = new URLRequest(url);

// Lanzamos la descarga
loader.load(urlImage);
```

- Afegir l'esdeveniment per al *loader* i el seu posterior funcionament per tal de *monotoritzar* la descàrrega.

```
loader.contentLoaderInfo.addEventListener(Event.COMPLETE, onComplete);
```

- Per últim crear la funció que hem emprat en l'esdeveniment anterior

```
//ETAPA 7 - CARGAS DE LA IMAGEN
public function progressHandler(e:ProgressEvent):void {
        trace("Percentatge descarregat : " + (e.bytesLoaded/e.bytesTotal)*100 + "%");
}
```

SEGONA PART

Etapa1 | Classe *ArrowBt*

Atès que les fletxes de navegació lateral són simètriques aprofitem un sol símbol gràfic "*arrow_sp*" que associem a una única classe *ArrowBt* (classe que trobareu ja creada i associada al seu símbol).
Dins de *AnimateCC* entreu en el símbol *arrow_sp* i observeu com està creat. Veureu que es tracta de 3 objectes superposats un sobre l'altre:

- la instància *normalState_sp*: que serà el seu aspecte normal
- la instància *hoverState_sp*: que serà el seu aspecte en *hover* (quan el ratolí sobrevola el botó)
- la instància *clicState_sp*: que serà el seu aspecte quan es fa clic sobre ell

Haureu de modificar la classe *ArrowBt* per a que escolti els esdeveniments adequats que permetin detectar quan el ratolí sobrevola la fletxa, deixa de sobrevolar-la o hi fa clic. En cada cas caldrà fer que sigui visible només un dels 3 objectes.
Important: Dins de la classe *ArrowBt* només ens ocupem d'aquesta interacció. No ens ocuparem de res relacionat amb el canvi d'imatge a la galeria, això es farà després, a la classe *Gallery*.

– Afegim els 3 esdeveniments per el *mouseClick*, *mouseHover*, *mouseOut* que seran al fer *click*, al passar per sobre i al sortir del botó.

```
        // AFEGIM EVENTS DEL RATOLI
        addEventListener(MouseEvent.CLICK, mouseClick);
        addEventListener(MouseEvent.MOUSE_OVER, mouseHover);
        addEventListener(MouseEvent.MOUSE_OUT, mouseOut);

        // Dado que ArrowBt hereda de Sprite
        // pero queremos tratarlo como un botón, nos interesa:
        // que el cursor cambie su aspecto al pasar por encima
        buttonMode = true;
        // que sus elementos internos queden inhabilitados para interacciones con el ratón
        mouseChildren = false;
    }

    private function mouseHover(ev:MouseEvent):void {
        //"ELIMINEM" INVADER
        trace("Mouse a sobre");
    }

    private function mouseOut(ev:MouseEvent):void {
        //"ELIMINEM" INVADER
        trace("Mouse out");
    }

    private function mouseClick(ev:MouseEvent):void {
        //"ELIMINEM" INVADER
        trace("Mouse click");
    }
```

- Ara un cop dins de cada funció, tenim que donar visibilitat a cada un de les instàncies

```
private function mouseHover(ev:MouseEvent):void {
    //"ELIMINEM" INVADER
    trace("Mouse a sobre");
    //hoverState_sp:Sprite;
    normalState_sp.visible = false;
    clickState_sp.visible = false;
    hoverState_sp.visible = true;
}

private function mouseOut(ev:MouseEvent):void {
    //"ELIMINEM" INVADER
    trace("Mouse out");
    normalState_sp.visible = true;
    clickState_sp.visible = false;
    hoverState_sp.visible = false;
}

private function mouseClick(ev:MouseEvent):void {
    //"ELIMINEM" INVADER
    trace("Mouse click");
    normalState_sp.visible = false;
    clickState_sp.visible = true;
    hoverState_sp.visible = false;
    //clickState_sp:Sprite;
}
```

Etapa2 | Classe *PhotoInfoPanel*

Haureu de completar la classe *PhotoInfoPanel*:

a) Crear el mètode públic *setInfo* ()

Aquest mètode rebrà dos paràmetres: títol de la foto i autor. Els guardarà en les seves variables privades _photoTitle i _photoAuthor respectivament; i refrescarà els camps de text que es veuen en el panell amb els nous valors.

- Hem realitzat malament en la crida de paràmetres ja que ens ha estat impossible donar el títol i el autor de les imatges, cosa que sí que hem pogut realitzar amb les imatges fácilment.

b) Afegir certa interactivitat.

A l'hora de visualitzar fotos ens interessa poder fer desaparèixer la informació de la foto, deixant només el botó *btInfo_bt* sempre visible. D'aquesta manera podrem visualitzar millor la imatge de la galeria. El botó *btInfo_bt* haurà de comportar-se doncs com un botó de tipus palanca (*toggle*). Fent clic sobre ell la informació (*photoTitle_tf*, *authorName_tf* i *btInfo_bt*) es farà visible o invisible.

- Per a la realització d'aquest apartat, hem realitzat una funció nova anomenada *btInfo_bt_Interactivity* on al fer *click* al botó *btInfo_bt* amagarà o mostrarà els paràmetres que ens demana l'anunciat.
- Hem creat una variable booleana anomenada "*activado*" on segons el seu valor mostrarem o no les propietats que ens demana l'anunciat (*photoTitle_tf, authorName_tf*, *photoInfoBg_sp*) i canviarem el valor de la variable "*activado*" per tal de que si tornem a clicar el botó, faci la inversa que va realitzar anteriorment (creant així com un "*toggle*").

```
        //ETAPA 2 - BOTO btInfo_bt
        btInfo_bt.addEventListener(MouseEvent.CLICK, btInfo_bt_Interactivity);
        //setInfo();
    }

    // ----------------------------------------------------------------------------------------------
    //ETAPA 2 - BOTO btInfo_bt VISIBLE O NO VISIBLE
    public function btInfo_bt_Interactivity(event:MouseEvent):void{
        if(activado= true){
            //NO FUNCIONA POSANT VISIBLE = TRUE O VISIBLE = FALSE; per això feim !photoTitle_TF.visible
            //es el mateix que fer !true o !false
            photoTitle_tf.visible = !photoTitle_tf.visible;
            authorName_tf.visible = !authorName_tf.visible;
            photoInfoBg_sp.visible = !photoInfoBg_sp.visible;
            activado = false;
        }else if(activado=false){
            photoTitle_tf.visible = true;
            authorName_tf.visible = true;
            photoInfoBg_sp.visible = true;
            activado = true;
        }
    }
```

Etapa3 | Classe Gallery, navegació foto anterior i foto posterior

La classe *Gallery* controlarà la navegació per la nostra galeria. En tot moment la seva variable *_actualSlideNumber* ens indicarà la imatge de la galeria que estem visualitzant. D'altra banda la seva variable *_numSlides* guardarà el nombre d'imatges de la qual està composta la galeria.

En aquesta etapa ens ocuparem de la navegació amb les fletxes laterals, sense entrar encara en el tema de descàrrega de la imatge.

1.**Modificar la classe *Gallery* per a que:**

En fer clic amb el ratolí sobre la fletxa dreta s'executi el mètode privat *nextSlide*(). Haureu de crear també aquest mètode, però per ara simplement executarà un simple trace: trace("Foto següent") i modificarà convenientment la variable *_actualSlideNumber*

Fer el mateix per a la fletxa esquerra. En fer clic amb el ratolí sobre la fletxa esquerra haurem d'executar el mètode privat *prevSlide*(), que fa un trace: trace("Foto anterior") i modifica la variable *_actualSlideNumber*.

Un cop actualitzada la variable *_actualSlideNumber*, tant a *nextSlide*() com a *prevSlide*(), executareu el mètode *updatePhotoInfo*(). Aquest mètode, que haureu de crear, aprofitarà *_actualSlideNumber* per recuperar les noves dades, títol i autor de la foto, de *_galleryData* i refrescar els camps del panell d'informació.

Per acabar haureu de fer les modificacions convenients perquè en cas d'estar a la primera imatge de la galeria no aparegui la fletxa esquerra (ja que no hi ha imatge precedent) i quan estem a l'última imatge no aparegui la fletxa dreta (ja que no existeix imatge posterior).

- Hem creat les funcions *nextSlide*() i *prevSlide*() i inclòs el "*trace*" corresponent que ens demana l'anunciat. Cadascun d'aquest, executarà la funció *updatePhotoInfo()* per tal de procedir amb el tractament de les imatges a mostrar per pantalla.
- Cal dir que en les funcions *nextSlide*() i *prevSlide*() tractarem el tema d'eliminar les imatges anteriors o posteriors que ens demana en la etapa 6.
- També hem tractat el tema de les fletxes per tal de que en la primera imatge no es vegi la fletxa de la esquerra i en la última no es vegi la fletxa de la dreta. Això ho **hem realitzat en el** *updatePhotoInfo*().

```
private function nextSlide(ev:MouseEvent = null):void {
    trace("Foto següent ");

    //trace(_actualSlideNumber);
    //TRACE PER EL SLIDENUMBER
    _actualSlideNumber = _actualSlideNumber+1;

    //ELIMINEM ELS CHILDS ANTERIORS I AFEGIREM ELS SEGUENTS POSTERIORMENT AMB UPDATEPHOTOINFO
    if(_actualSlideNumber == 1){
        photoContainer_sp.removeChildAt(_galleryData[0].imageUrl);
    }else if(_actualSlideNumber == 2){
        photoContainer_sp.removeChildAt(_galleryData[1].imageUrl);
    }else if(_actualSlideNumber == 3){
        photoContainer_sp.removeChildAt(_galleryData[2].imageUrl);
    }

    //UPDATE PHOTO INFO
    updatePhotoInfo();
}

private function prevSlide(ev:MouseEvent = null):void {
    trace("Foto anterior ");

    //photoContainer_sp.removeChildAt(_actualSlideNumber);
    //TRACE PER EL SLIDENUMBER
    _actualSlideNumber = _actualSlideNumber-1;

    //ELIMINEM ELS CHILDS ANTERIORS I AFEGIREM ELS SEGUENTS POSTERIORMENT AMB UPDATEPHOTOINFO
    if(_actualSlideNumber == 2){
        photoContainer_sp.removeChildAt(_galleryData[3].imageUrl);
    }else if(_actualSlideNumber == 1){
        photoContainer_sp.removeChildAt(_galleryData[2].imageUrl);
    }else if(_actualSlideNumber == 0){
        photoContainer_sp.removeChildAt(_galleryData[1].imageUrl);
    }

    //UPDATE PHOTO INFO
    updatePhotoInfo();
}
```

- Anem indicant la imatge corresponent gràcies a introduir-hi el *_galleryData[X]* que volem que ens faci en cada etapa.
- Al mateix temps, crearem les diferents opcions que ens demana per la funció *updatePhotoInfo()* on actualitzarem segons el _actualSlideNumber tots els paràmetres.
- **NOTA**: Al no haver realitzar l'etapa 2, els paràmetres de "*title*" o "*author*" no les empram per res en aquest apartat.

```
private function updatePhotoInfo(ev:MouseEvent = null):void {
    // Loader nos permite cargar una imagen y monitorizar su descarga
    var loader:Loader = new Loader();

    //VARIABLES RANDOM PER LACTUALITZACIO DE DADES
    var url:String;
    var title:String;
    var autor:String;

    //ACTUALITZAM DADES
    url = _galleryData[_actualSlideNumber].imageUrl;
    title = _galleryData[_actualSlideNumber].title;
    autor = _galleryData[_actualSlideNumber].author;

    // Creamos el urlRequest para utlizar con loader
    var urlImage:URLRequest = new URLRequest(url);
    loader.load(urlImage);
    //photoContainer_sp.removeChildAt(_actualSlideNumber-1);
    loader.contentLoaderInfo.addEventListener(Event.COMPLETE, onComplete);


    //TRACTAMENT PER ELS ARROW (prev o next)
    if(_actualSlideNumber != 0 && _actualSlideNumber != 3){
        arrowL_sp.visible = true;
        arrowR_sp.visible = true;
    }else if (_actualSlideNumber == 0){
        arrowL_sp.visible = false;
    }else if(_actualSlideNumber == 3){
        arrowR_sp.visible = false;
    }

    //ETAPA 7
    loader.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS, progressHandler);
    //PRELOADER ETAPA 7
    preloader = new PreloaderBar();

    // AFEGIM EL PRELOADER
    preloaderStart();
}
```

Etapa4 | Classe *Gallery*: control via teclat

Aprofitant que la nostra aplicació podria executar-se com a aplicació d'escriptori voldríem permetre navegar a la foto anterior i posterior gràcies a les fletxes esquerra i dreta del teclat, respectivament. Haureu de fer els canvis convenients per a que sigui possible.

- Per el control de teclat, hem creat les següents línies de codi:

```
//ETAPA 4 - CONTROL AMB KEYBOARD
function myKeyDown(e:KeyboardEvent):void{
    //Afegim un control per no sobrepassar les imatges ja que donaría error, per això
    //direm que NO ha de ser més gran a 3
    if(e.keyCode == 39 && _actualSlideNumber<3){
        nextSlide();
    //Afegim un control per no sobrepassar les imatges ja que donaría error
    //direm que ha de ser DIFERENT a 0
    }else if(e.keyCode ==37 && _actualSlideNumber!=0){
        prevSlide();
    }
}
```

Etapa5 | Classe *Gallery*: control tàctil

Finalment i atès que també podrem executar l'aplicació en dispositius tàctils afegirem la possibilitat de passar d'una foto a la següent o anterior fent un *swipe* lateral, d'esquerra a dreta per passar a la foto anterior i de dreta a esquerra per passar a la foto posterior. Haureu de fer els canvis convenients per a que sigui possible.

- Per l'etapa 5 hem creat les següents línies de codi, a part de introduir tots els imports corresponents:

```
function onSwipe (e:TransformGestureEvent):void{
    //trace("AITORRRRRMENTA");
    if (e.offsetX == 1) {
        //User swiped towards right
        stage.x += 100;
    }
    if (e.offsetX == -1) {
        //User swiped towards left
        stage.x -= 100;
    }
    if (e.offsetY == 1) {
        //User swiped towards bottom
        stage.y += 100;
    }
    if (e.offsetY == -1) {
        //User swiped towards top
        stage.y -= 100;
    }
}
```

Etapa6 | Classe *Gallery*: mètode *loadPhoto*()

Ara ens ocuparem de descarregar i visualitzar la imatge actual.
Haureu d'afegir el codi adequat al mètode *loadPhoto*() per a que gràcies a *_actualSlideNumber* recuperi la *url* de la imatge i la descarregui. Un cop descarregada la imatge s'afegirà a la *displayList* de *photoContainer_sp*, que ja està creat.
Lògicament caldrà afegir a *nextSlide*() i *prevSlide*() l'execució de *loadPhoto*() després d'haver actualitzat el valor de *_actualSlideNumber*.
També caldrà tenir en compte que si existeix ja una imatge visible abans caldrà eliminar-la.

```
private function loadPhoto():void {
    trace("Càrrega de fotografia nº "+_actualSlideNumber);

    // Loader nos permite cargar una imagen y monitorizar su descarga
    var loader:Loader = new Loader();

    var url:String;

    //image source can be online
    //note: adding random will make flash load image always as a new image, therefore not keeping it on cache
    url = _galleryData[0].imageUrl;

    // Creamos el urlRequest para utlizar con loader
    var urlImage:URLRequest = new URLRequest(url);

    // Lanzamos la descarga
    loader.load(urlImage);
    //loader.width = photoContainer_sp.width;
    //loader.height = photoContainer_sp.height;
    loader.contentLoaderInfo.addEventListener(Event.COMPLETE, onComplete);
    //photoContainer_sp.addChild(loader);


    //INICI DELS BOTONS SI ACTUALSLIDE ES 0 NOMES VEREM EL NEXT
    if(_actualSlideNumber == 0){
        arrowL_sp.visible = false;
        arrowR_sp.visible = true;
    }

    // Añadimos el preloader, que situamos en el centro de nuestra aplicación
    //photoContainer_sp.addChild(preloader);

    //ETAPA 7
    loader.contentLoaderInfo.addEventListener(ProgressEvent.PROGRESS, progressHandler);
    loader.contentLoaderInfo.addEventListener(Event.COMPLETE, completeHandler);
    //PRELOADER ETAPA 7
    preloader = new PreloaderBar();
}
```

- Podem veure com introduïm les imatges en el *loadPhoto*(). També podem veure c**om hem tractat el tema de eliminar les imatges**, les quals les hem realitzat en les etapes del *prevSlide* i *nextSlide*.

Etapa7 | Classe *Gallery*: *monitorització* de la descàrrega

Quan descarreguem fitxers externs (com poden ser imatges) depenent del pes del fitxer i de la connexió a Internet que tinguem, podem estar esperant un cert temps. En aquest temps d'espera un usuari que utilitzi l'aplicació pot tenir la impressió que "no fa res" i pensar que ha deixat de funcionar. És per això essencial poder *monitoritzar* la descàrrega del fitxer i informar a l'usuari del que passa.
Haureu de modificar la classe *Gallery* per a que al carregar una nova imatge *monitoritzi* la descàrrega d'aquesta imatge.
En aquesta etapa n'hi haurà prou fent un trace del % descarregat de la imatge, acabant amb un trace que indiqui "Descàrrega completada".

- Per fer aquest exercici hem emprat el que ens donaven com a exemple amb la PAC2, així el codi queda similar, tal com podem veure a continuació (al mateix temps, apareix el codi per el percentatge per l'apartat posterior).
- Cal dir que també tenim que introduir el "*listeners*" corresponents dins del *loadPhoto*().

```
//ETAPA 7 - CARGAS DE LA IMAGEN
public function progressHandler(e:ProgressEvent):void {
    trace("Percentatge descarregat : " + (e.bytesLoaded/e.bytesTotal)*100 + "%");
    preloader.percentage((e.bytesLoaded/e.bytesTotal)*100);
}
private function completeHandler(e:Event):void {
    trace("Descàrrega completada");
}
```

Etapa8 (OPCIONAL) | Classe *Gallery*: ús de *preloader* per informar l'usuari

Atès que l'ús de trace només és útil en desenvolupament, per indicar a l'usuari de *l'ARM* final com transcorre la descàrrega d'una foto utilitzarem un *preloader*.
Aprofitant el símbol *preloader_mc* i la seva classe ja associada *PreloaderBar* haureu de mostrar durant la descàrrega d'una imatge el percentatge descarregat.
Un cop la imatge descarregada el *preloader* desapareixerà.

- Juntament amb les línies de codi anterior (del *progressHandler*), on posem el *preloader* juntament amb les línies de a continuació (i altres declaracions) realitzem aquest exercici.

```
//------------------------------------------------------------------------------------------
//ETAPA 8 - PRELOADER
private function preloaderStart(){
    preloader.x = (stage.stageWidth-preloader.width)/2;
    preloader.y = (stage.stageHeight-preloader.height)/2;
    photoContainer_sp.addChild(preloader);
}
//------------------------------------------------------------------------------------------
```

Etapa9 (OPCIONAL) | Classe *Gallery*: ajustar imatge

La imatge descarregada i afegida a la *displayList* de *photoContainer_sp* apareixerà en la seva grandària original, però normalment s'hauria d'adaptar a la pantalla del nostre dispositiu. Depenent de les proporcions de la imatge i del dispositiu això pot significar ajustar la imatge a l'ample de la pantalla o bé ajustar l'alçada de la imatge a l'alçada de la pantalla.
Fer els canvi corresponents per a que sempre s'ajusti el millor possible, quedi centrada tant horitzontal com verticalment i veiem la imatge el més gran possible sense sortir de la pantalla.

- Apliquem aquest codi de a continuació per tal de realitzar aquesta etapa. La clau està en emprar *stage.stageWidth* i *stage.stageHeight* en lloc de simple *stage.widht* o *stage.height*.

```
//ETAPA 9 OPCIONAL -- IMATGE A LA TOTAL GRANDARIA
public function onComplete(e:Event):void {
    var img:Bitmap = Bitmap(e.target.content);
    photoContainer_sp.addChild(img);

    //stage.stageWidth | stage.stageHeight per la mida del stage
    img.width = stage.stageWidth;
    img.height = stage.stageHeight;
}
```

Bibliografia

- GÓNZALEZ SANCHO, MARCOS. (2016). "*Introducció i conceptes generals*". [en línia]. Catalunya: Universitat Oberta de Catalunya.
  - http://materials.cv.uoc.edu/continguts/PID_00192290/web/main/m1/portada.html
- GÓNZALEZ SANCHO, MARCOS. DE FUENMAYOR LÓPEZ, DANIEL. (2016). "*Cicle de desenvolupament d'una aplicació Rich Media*". [en línia]. Catalunya: Universitat Oberta de Catalunya.
  - http://materials.cv.uoc.edu/continguts/PID_00192290/web/main/m2/portada.html
- DE FUENMAYOR LÓPEZ, DANIEL. (2016*). "Desenvolupament d'aplicacions Rich Media en la Plataforma Flash*". [en línia]. Catalunya: Universitat Oberta de Catalunya.
  - http://materials.cv.uoc.edu/continguts/PID_00192290/web/main/m3/portada.html
- GÓNZALEZ SANCHO, MARCOS. (2016). *"Publicació i model de negoci".* [en línia]. Catalunya: Universitat Oberta de Catalunya.
  - http://materials.cv.uoc.edu/continguts/PID_00192290/web/main/m4/portada.html
- GÓNZALEZ SANCHO, MARCOS. (2013). *"Tecnologías para desarrollo de aplicaciones rich media"*. [en línia]. Catalunya: Universitat Oberta de Catalunya.
  - http://mosaic.uoc.edu/2013/07/31/tecnologias-para-desarrollo-de-aplicaciones-rich-media/#
- MONCHO MAS, VICENT. (2016). "*Introducció a la programació orientada a objectes*". [en línia]. Catalunya: Universitat Oberta de Catalunya.
  - http://materials.cv.uoc.edu/continguts/PID_00220463/index.html

Recursos Multimèdia

- Imatge corporativa de la Universitat Oberta de Catalunya. "*UOC logotipo azul papel*". [Imatge]. http://www.uoc.edu/portal/es/universitat/coneix/marca/logotip-paper/index.html
- La resta d'imatges són de creació pròpia.