1 Klassen per module

Hierin worden alle grote klassen beschreven per module die gebruikt worden in de IDE. Elke klasse wordt kort toegelicht over hun functie, welke ADT's gebruikt hebben alsook hoe alle klassen samenwerken.

1.1 Variables

De verantwoordelijkheden van deze module worden beschreven in Sectie **??**.

1.1.1 EventInstance

Een event bevat een Event voor zijn type Event waarvan het een instance is bij te houden en een Hashmap [**?**] waarbij het de naam van een variable (String) mapt op een variabele.

1.1.2 ReturnVariables

De ReturnVariables is klasse die gebruikt om return values in op te slaan en terug aan de functioncall. Deze bevat een stack [**?**] waarop de return waarde worden geduwt. Op deze manier kunnen we meerdere return values mogelijk maken. Echter voorzien we eerst maar een return value.

1.1.3 Type

Dit is Enumeratie die een van de volgende kan zijn: Boolean, String, Number, Event of Value. Een value is een letterlijke inputwaarde door de gebruiker. Deze is nog niet geconverteerd naar een feitelijk type en kan dus eenderwelk type voorstellen.

1.1.4 Abstact Class Variable

Deze klasse stelt een variabele voor. Deze heeft een Type als member.

1.1.5 BooleanVariable

Deze wordt afgeleid van de Abstact Class Variable en bevat dus een boolean als variable.

1.1.6 NumberVariable

Deze wordt afgeleid van de Abstact Class Variable en bevat dus een number als variable.

1.1.7 StringVariable

Deze wordt afgeleid van de Abstact Class Variable en bevat dus een String als variable.

1.2 Blocks

De verantwoordelijkheden van deze module worden beschreven in Sectie **??**.

1.2.1 De Interface Block

Een Block stelt een klasse voor die een bepaald stuk code voorstelt. Deze bevat een execute functie die een proces meekrijgt zodat hij zijn code kan uitvoeren of plaaten op het proces.

1.2.2 HandlerBlock

Deze implementeert de interface Block. Deze bevat een `ArrayList<Block>` [**?**] die we de body noemen. De heeft een EventInstance die hij mee kreeg op oproep, via zijn constructor. Deze wordt mee op zijn variabele stack gepushed bij het aanmaken van zijn FunctionFrame. De execute van deze block pushed de body als individuele blokken op de stack van het proces dat hij meekrijgt. Achteraan de body plakt hij nog een PopBlock.

1.2.3 PopBlock

Deze implementeert de interface Block. De execute van deze Block popt het bovenste FunctionFrame van de stack van het proces.

1.2.4 AccessBlock

Deze bevat twee Strings namelijk de naam van het EventInstance waarvan het een member wil opvragen. En de naam van die member. De execute functie zal dus het EventInstance van de FunctionFrame opvragen. Hierin vraagt hij de variable op van de member en deze geeft hij terug.

1.2.5 BroadCastBlock

Bevat een naam van het Event. En een Hashmap [**?**] van Strings die de members van het event voorstellen deze worden gemapt op Blocks. De execute van dit Block zal deze Blocks uitvoeren en de bekomen variabele kopieren en mappen op de juiste string. Het zal dan een aangemaakte event terug geven aan het proces. Dat dit op zijn beurt doorgeeft aan de VM.

1.2.6 FunctionBlock

Deze implementeert de interface Block. Deze bevat een `ArrayList<Block>` [**?**] die we de body noemen. De excute functie van deze Block zet de body op Stack van Blocks van het proces dat deze meekrijgt. Onder deze body plakt hij nog PopBlock. De Block bevat twee `ArrayList<VariableBlock>` [**?**] voor respectievelijke parameters en return parameters.

1.2.7 FunctionCallBlock

Deze bevat een String voor de functie naam. En twee `ArrayList<String>` [**?**] voor respectievelijk de parameters en return waardes van de Call.
De execute van deze Block zal de variabele van de waarde parameters ophalen uit het huidige bovenste FunctieFrame van de Stack van het proces. Deze slaat hij tijdelijk lokaal op. Hierna haalt hij de namen van de parameters van de functie op. Hij maakt een nieuwe FunctieFrame hierop pushed hij alle parametersnamen met de eerder opgehaalde variabelen. Hierna pushed hij op Stack de SetBlocken voor de return waardes. Uiteindelijk roept hij de execute van de functie aan zodat deze bovenaan de stack staat. Dit telt als een primitieve stap.

1.2.8 ReturnBlock

Deze block bevat een `ArrayList` [**?**] van Strings die de namen van de variables voorstellen die gereturned moeten worden. Deze zal hij ophalen in het huidige FunctieFrame en opslaan in de ReturnVariables van het proces. Een ReturnBlock popt alle blokken van de Stack tot hij een PopBock tegenkomt.

1.2.9 VariableBlock

Deze Block bevat een String en een Type. De execute van deze block zal deze variable aanmaken en op het huidige FunctionFrame zetten.

1.2.10 SetBlock

Deze bevat een String die de naam is van een variabele op de huidige FunctionFrame. Deze block bevat nog een andere block, dat ofwel een variable, value of operatie aanduid. De execute van eender van deze blocken geeft steeds de inhoud (variable) terug aan de SetBlock. Deze blok telt als een primitieve stap.

1.2.11 ArithBlock

Deze block bevat een left block en een right block. Als deze twee worden geexecute op de teruggeven variable wordt de juiste operatie uitgevoerd de bekomen variabele wordt terug gegeven. De block bevat ook een statische hashmap zoals beschreven in sectie **??** om de juiste operatie uit te kunnen voeren. De uitvoering zal dus in een primitieve stap gebeuren.

1.2.12 Random

De execute van deze block geeft een random value tussen een lower- en upperBound terug in een Variabele. De lower- en upperBound kunnen weer blokken zijn die worden execute en een variabele teruggeven.

1.2.13 ConcatBlock

De execute van deze block geeft een variable terug die de string concatinatie van de een left- en rightBlock is. Deze Blocken kunnen dieper genest zijn maar hun execute geeft een variable terug.

1.2.14 StrlenBlock

Deze block geeft een variable terug die de lengte van de String bevat. Het bevat een Block die op zijn execute een Stringvariable terug geeft.

1.2.15 CharAtBlock

Deze block geeft een variable terug die een string op een gegeven index. De String wordt meegeven als een block en deze geeft dus een variabele terug. De block die de string bevat kan dus genest zijn. Als die index niet gevonden is, dan wordt er een OutOfBoundsException gegooid.

1.2.16 LogicBlock

De execute van deze block geeft een variable terug. Deze block bevat een left block en een right block. Als deze twee worden geexecute op de teruggeven variable wordt de juiste operatie uitgevoerd de bekomen variabele wordt terug gegeven. De uitvoering zal dus in een primitieve stap gebeuren. De block bevat ook een statische hashmap zoals beschreven in sectie **??** om de juiste operatie uit te kunnen voeren.

1.2.17 LockBlock

Deze block lockt een bepaalde membervariabele van de instance waarbij het huidige proces hoort.

1.2.18 UnlockBlock

Deze block unlockt een bepaalde membervariabele van de instance waarbij het huidige proces hoort.

1.2.19 ForeverBlock

Deze block bevat een `ArrayList<Block>` [**?**] die we body noemen. De execute van deze block zet al zijn blokken op de stack met onderaan de body ook nog eens zichzelf geplakt.

1.2.20 WhileBlock

Deze bevat een Block conditie en een `ArrayList<Block>` [**?**] die we body noemen. De execute van de block kijkt als de conditie naar true evalueert door deze te laten execute en de waarde van de variable na te kijken. Zo ja dan wordt de body plus zichzelf op de stack van het proces gepushed.

1.2.21 IfElseBlock

Deze block bevat een Block conditie en twee `ArrayList<Block>` [**?**] die respecitievelijk de body van de If en else voorstellen. De execute van de block kijkt als de conditie naar true evalueert door deze te laten execute en de waarde van de variable na te kijken. Zo ja dan wordt de body van de If op de stack van het proces gepushed anders de body van de Else.

1.2.22 HideBlock

Voert een functie van de instance uit om te hiden.

1.2.23 ShowBlock

Voert een functie van de instance uit om te showen.

1.2.24 ChangeAppereanceBlock

De bevat een Block index, deze kan een arith expression zijn. Dus we laten hem execute zodat we de index krijgen in een variable. Dit zal de execute van de blok doen plus het oproepen van de functie bij een instance die de appereance veranderd.

1.2.25 MoveBlock

Deze bevat een x- en een y-Block. De execute van de MoveBlock zal de waarde van x en y bekomen door deze Blocks te execute. De waardes geeft hij mee aan een functie van de instance die zijn x en y verhoogt met die waardes.

1.2.26 PrintBlock

Deze bevat een Block die geexecute wordt en de waarde van de variable wordt uitgeprint.

1.3 Core

De verantwoordelijkheden van deze module worden beschreven in Sectie **??**.

1.3.1 Instance

Deze klasse stelt een instantie van een Class voor. Deze bevat een HashMap [**?**] van variables deze zijn zijn member variables maar ook zijn positie. Een instance bevat ook zijn Class zodat men weet welke events en functies deze bevat. Een instance bevat ook een HashSet van zijn locked members.

1.3.2 EventDispatcher

De EventDispatcher krijgt een eventInstance en een afzender van de virtual machine. De EventDispatcher kent alle wires en instances. Hij zal voor het gegeven eventInstance alle handlers van de ontvangers opvragen die voor dat event met de afzender zijn verbonden. Voor elke handler creeërt hij een nieuw proces dat hij aan de virtual machine geeft.

1.3.3 Proces

Het proces bevat een stack van Blocks. Initiël bevat die stack enkel de handler die opgeroepen is. Het bevat de instantie waarvan het de handler uitvoert. Het bevat ook een Stack van functionFrames (van Hashmaps waarbij Strings worden gemapt op Variables (zie Sectie **??**)(zie Sectie **??**).
Deze laatste stack stelt de stack van actieve functies voor.
Een proces bevat een run-functie die één primitieve stap zal uitvoeren. Deze functie kan een event teruggeven als deze gecreeërd werd in de primitieve stap. Als het geen event terug geeft is er geen event gecreeërt maar is het proces nog niet afgelopen. Als het wel een afgelopen (de stack van Blocks is leeg) zal het een ProcesFinishedException gooien. Hierdoor weet de virtualmachine dat het event niet terug op de queue moet worden gezet. Het proces bevat ook nog een klasse ReturnVariables.
De klasse bevat ook nog een boolean locked die aangeeft als het proces verantwoordelijk is voor het locken van een variabele. Hierop kunnen er wel nog nieuwe locks gebeuren binnen dat proces. Een voorbeeld van de werking van een proces is uitgewerkt in Sectie **??**.

Keuze ADT's: De Stack [**?**] van Blocks zorgt ervoor dat telkens de bovenste blokken geexecute kunnen worden. De functie wordt er dus telkens volledig bovenop gezet. Omdat we enkel bovenaan moeten toevoegen en uitvoeren gebeurt deze operatie in O(1). We gebruiken hiervoor `java.util.Stack<Block>` heb Voor informatie over het toevoegen van de blokken van een functie op deze stack verwijzen we naar de Class FunctieBlock.

Het gebruiken van een Stack [**?**] zorgt ervoor dat we weten in welk blok er gezocht moet worden naar de lokale variabele van de huidige functie. Hierdoor kan de totale operatie van zoeken in O(1). Als een variable niet gevonden wordt, zoekt men in de member variables van de instance.

1.3.4 VirtualMachine

De virtual machine bevat een lijst (queue) van processen (zie Sectie **??**)en de eventDispatcher. Hij doorloopt de lijst van processen door telkens de voorste eraf te halen. Hij zal elk proces één primitieve stap laten uitvoeren. Als het proces in die primitieve stap een event creeërt zal de VM (virtual machine) dit event en zijn zender doorgeven aan de eventDispatcher. Zoals eerder vermeldt zal de eventDispatcher een hoeveelheid processen teruggeven (`ArrayList<Proces>`

[**?**]). Deze worden achteraan de lijst toegevoegd. Als het huidige proces nog niet is afgerond wordt dit ook achteraan terug toegevoegd (na de eventuele nieuwe processen).

Keuze ADT's: Voor de queue gebruiken we `java.util.LinkedList<Proces>` [**?**] hierdoor kan het afnemen van het eerste element en toevoegen van een element aan de queue in O(1). Natuurlijk is de orde van belang in de queue wat hier FIFO is.

1.3.5 FunctionFrame

Een functionFrame stelt het stackframe voor van een functie blok. Deze bevat een hashmap [**?**] waarbij de lokale Variables van die functie gemapt worden op hun naam in die functie.

Keuze ADT's: Een Hashmap waarbij Strings worden gemapt op variables. De keuze van een Hashmap zorgt ervoor dat het toevoegen van nieuwe variabelen en het opzoeken van bestaande in O(1) tijd kan gebeuren aangezien een functie een beperkt aantal lokale variabele bevat. Door het gebruik van een Hashmap is er ook geen compile stap nodig omdat we de variable dadelijk kunnen mappen op de content en geen indices van een array moeten worden berekent.[**?**]