Assignment 1
First & Follow; Recursive Descent Parser Proof

First & Follow

Production Rule	Non-Terminal	FIRST	FOLLOW
Rule1 `<jClass > ::= <className> B <varlist> {<method>} E`	<JClass>	FIRST(<className>) = {C,D}	Start Symbol Rule ends with implicit '$' = {'$'}
Rule2 `<className> ::= C|D`	<className>	{C,D}	(from Rule1) {B}
Rule3 `<varlist> ::= <vardef> {, <vardef>};`	<varlist>	FIRST(<vardef>) = FIRST(<type>) = {I,S}	(from Rule1) FIRST(<method>) U (from Rule1 if no method) {E} = {P,E}
Rule4 `<vardef> ::= <type> <var>`	<vardef>	FIRST(<type>) = {I,S}	(from Rule3 first occurrence of vardef) FIRST({, <vardef>}) U (from Rule3 second occurrence of vardef) {;} U (from Rule7 second occurrence of vardef {). Remember first occurrence is similar to first occurrence in Rule3} = {',', ';',')'}
Rule5 `<type> ::= I | S`	<type>	{I,S}	(from Rule4) FIRST(<var>) U (from Rule7) FIRST(<mname>) = {V,Z,M,N}
Rule6 `<var> ::= V|Z`	<var>	{V,Z}	(from Rule4 follow(vardef)) (from Rule11 and Rule13) {'=','==', ',', ';'}
Rule7 `<method> ::= P <type> <mname> (<vardef> {, <vardef>} ) B <stmnt> <returnstmnt> E`	<method>	{P}	(Rule1 method is followed by method in {method} AND IF NO METHOD IS followed by E) FIRST(<method>) U {E} = {P,E}
Rule8 `<mname> ::= M|N`	<mname>	{M,N}	(Rule7) {'('}
Rule9 `<stmnt> ::= <ifstmnt>|<assignstmnt>|`	<stmnt>	FIRST(<ifstmnt>) U FIRST(<assignstmnt>)	(Rule7) FIRST(<returnstmnt>) U

<whilestmnt>		U FIRST(<whilestmnt>) = {F,V,Z,W}	(Rule10 and 12: stmnt followed by stmnt in {stmnt}) {FIRST(<stmnt>)} U {E} = {R,F,V,Z,W,E}
Rule10 <ifstmnt> ::= F <cond> T B {<stmnt>} E [L B { <stmnt> } E]	<ifstmnt>	{F}	Same as Follow(stmnt) since ifstmnt has empty follow in rule9. FOLLOW(<stmnt>) = {R,F,V,Z,W,E}
Rule11 <assignstmnt> ::= <var> = <digit>;	<assignstmnt>	{V,Z}	Same as Follow(stmnt) since assignstmnt has empty follow in rule9. FOLLOW(<stmnt>) = {R,F,V,Z,W,E}
Rule12 <whilestmnt> ::= W <cond> T B <stmnt> {<stmnt>} E	<whilestmnt>	{W}	Same as Follow(stmnt) since whilestmnt has empty follow in rule9. FOLLOW(<stmnt>) = {R,F,V,Z,W,E}
Rule13 <cond> ::= (<var> == <digit>)	<cond>	{‘(‘}	(Rule12) {T}
Rule14 <returnstmnt> ::= R <var>;	<digit>	{0,1,2,3,4,5,6,7,8,9}	(Rule7) {E}
Rule15 <digit> ::= 0|1|2|3|4|5|6|7|8|9	<returnstmnt>	{R}	(Rule11 and 13) {‘;’,’)’}

See Proof for usability of Recursive Descent Parser for this Grammar in next page.

Proof to verify if the Recursive Descent Parsing technique can be used for above grammar.

Rule 1: Consider the Production Rules 5,6,8,9,16 which have the right hand side using '|' to separate alternate rules.

Applying the constraint rule on Page 7 of notes (05.Syntactic analysis and Predictive Parsing):

For <stmnt> ::= <ifstmnt>|<assignstmnt>|<whilestmnt>: FIRST(<ifstmnt>) ∩ FIRST(<assignstmnt>) ∩ FIRST(<whilestmnt>) = {F} ∩ {V,Z} ∩ {W} = {Θ}

For <digit> ::= 0|1|2|3|4|5|6|7|8|9, Clearly ∩ of separate parts = {Θ}

Same with Production Rules 5,6,8 which is obvious.

Rule 2:

No such rules apply here

Optional Part of an RHS (constraint rules on Page 8 of notes):

<ifstmnt> ::= F <cond> T B {<stmnt>} E [L B { <stmnt> } E]

Because [L B { <stmnt> } E] ends rule, it implies that it may end as λ, therefore:
FIRST([L B { <stmnt> } E]) ∩ FOLLOW(<ifstmnt>) = {L} ∩ {R,F,V,Z,W,E} = {Θ}

Indefinite Repeats

**For all cases object following all repeats do not generate λ, therefore
(Constraint rule on Page 8)**

<jClass > ::= <className> B < varlist> {<method>} E
FIRST(<method>) ∩ FIRST(E) = {P} ∩ {E} = {Θ}

<varlist> ::= <vardef> {, <vardef>};
FIRST{, <vardef>}) ∩ FIRST(';') = {','} ∩ {';'} = {Θ}

<method> ::= P <type> <mname> (<vardef> {,<vardef>}) B <stmnt> <returnstmnt> E
FIRST{, <vardef>}) ∩ FIRST(')') = {','} ∩ {')'} = {Θ}

<ifstmnt> ::= F <cond> T B {<stmnt>} E [L B { <stmnt> } E]
FIRST(<stmt>) ∩ FIRST(E) = {F,V,Z,W} ∩ {E} = {~~O~~}
This applies for both instances of {<stmnt>}.

<whilestmnt> ::= W <cond> T B <stmnt> {<stmnt>} E
FIRST(<stmnt>) ∩ FIRST(E) = {F,V,Z,W} ∩ {E} = {~~O~~}

Because Rules 1, 2, optional part of RHS and indefinite repeats were followed we can use predictive parsing.