

The background is a light gray gradient. It is decorated with numerous realistic water droplets of various sizes, some clustered in the top-left and bottom-right corners. A faint, circular, embossed-style logo is centered in the upper half of the image.

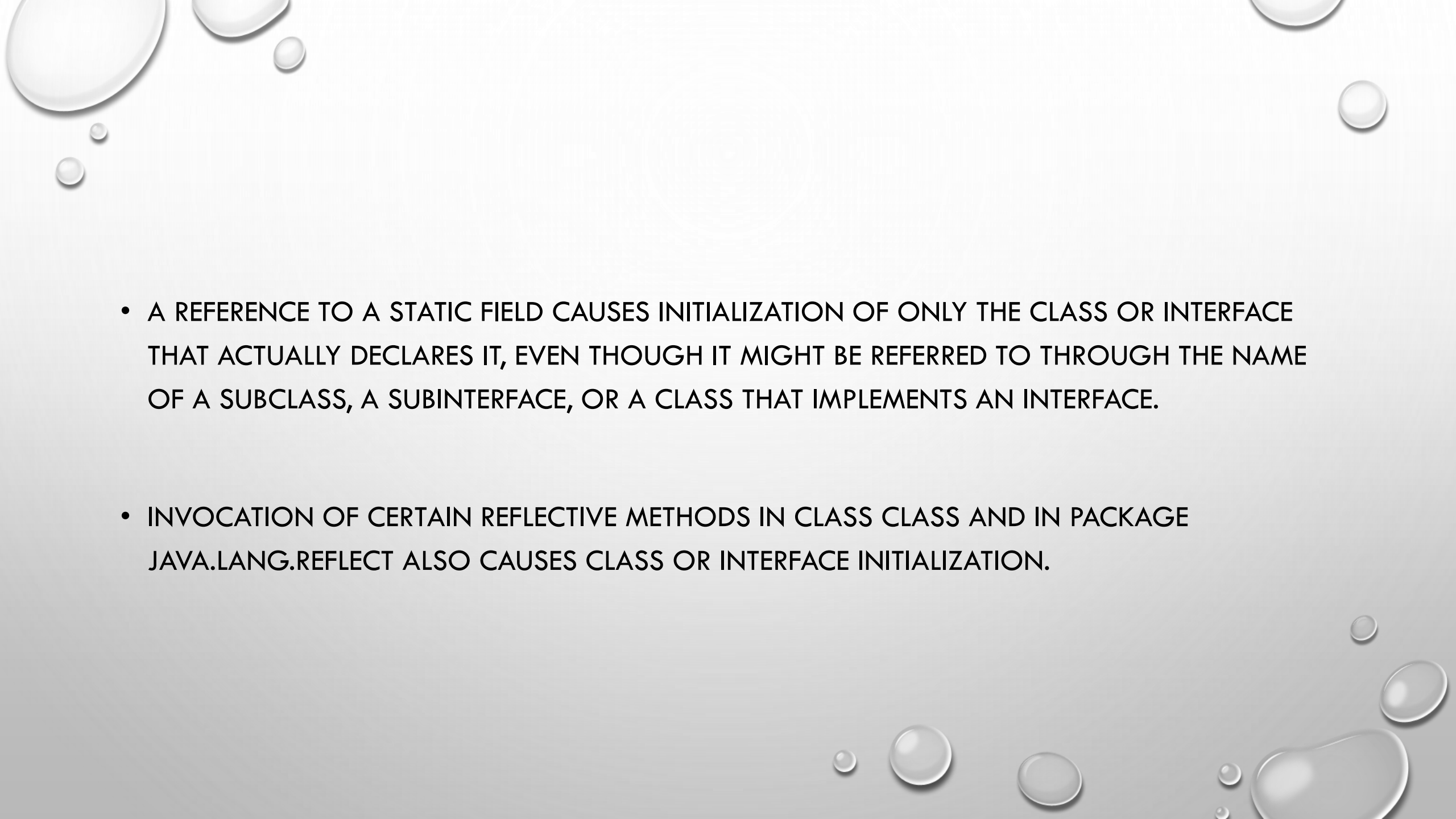
JAVA INITIALIZATION

INITIALIZATION OF CLASSES AND INTERFACES

- INITIALIZATION OF A CLASS CONSISTS OF EXECUTING ITS STATIC INITIALIZERS AND THE INITIALIZERS FOR STATIC FIELDS (CLASS VARIABLES) DECLARED IN THE CLASS.
- INITIALIZATION OF AN INTERFACE CONSISTS OF EXECUTING THE INITIALIZERS FOR FIELDS (CONSTANTS) DECLARED IN THE INTERFACE.
- BEFORE A CLASS IS INITIALIZED, ITS DIRECT SUPERCLASS MUST BE INITIALIZED, BUT INTERFACES IMPLEMENTED BY THE CLASS ARE NOT INITIALIZED. SIMILARLY, THE SUPERINTERFACES OF AN INTERFACE ARE NOT INITIALIZED BEFORE THE INTERFACE IS INITIALIZED.

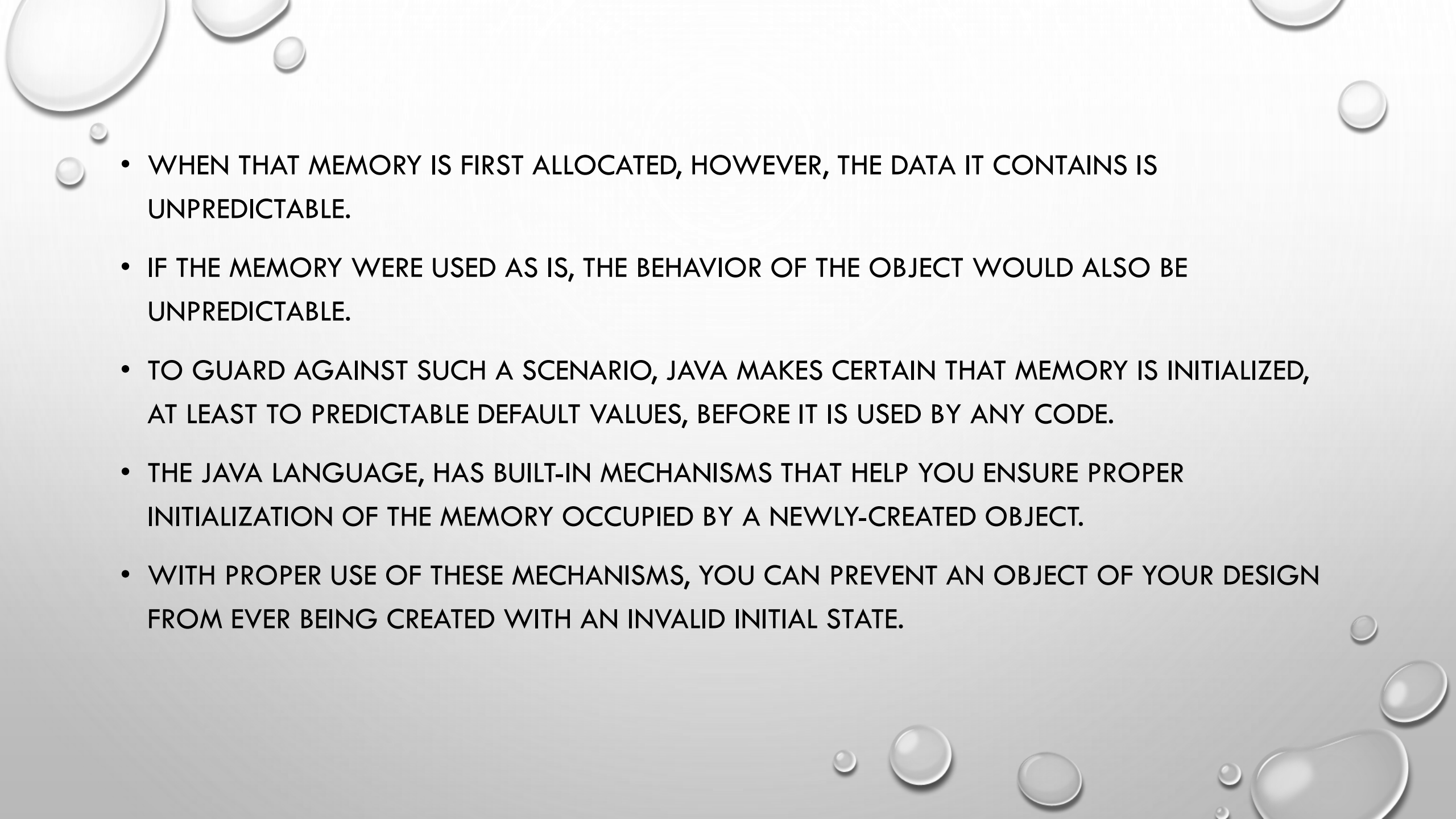
WHEN INITIALIZATION OCCURS

- A CLASS OR INTERFACE TYPE T WILL BE INITIALIZED IMMEDIATELY BEFORE THE FIRST OCCURRENCE OF ANY ONE OF THE FOLLOWING:
 - T IS A CLASS AND AN INSTANCE OF T IS CREATED.
 - T IS A CLASS AND A STATIC METHOD DECLARED BY T IS INVOKED.
 - A STATIC FIELD DECLARED BY T IS ASSIGNED.
 - A STATIC FIELD DECLARED BY T IS USED AND THE FIELD IS NOT A CONSTANT VARIABLE
 - T IS A TOP LEVEL CLASS, AND AN ASSERT STATEMENT LEXICALLY NESTED WITHIN T IS EXECUTED.

- 
- A REFERENCE TO A STATIC FIELD CAUSES INITIALIZATION OF ONLY THE CLASS OR INTERFACE THAT ACTUALLY DECLARES IT, EVEN THOUGH IT MIGHT BE REFERRED TO THROUGH THE NAME OF A SUBCLASS, A SUBINTERFACE, OR A CLASS THAT IMPLEMENTS AN INTERFACE.
 - INVOCATION OF CERTAIN REFLECTIVE METHODS IN CLASS CLASS AND IN PACKAGE JAVA.LANG.REFLECT ALSO CAUSES CLASS OR INTERFACE INITIALIZATION.

OBJECT INITIALIZATION IN JAVA

- AN OBJECT IS A CHUNK OF MEMORY BUNDLED WITH THE CODE THAT MANIPULATES MEMORY.
- IN THE MEMORY, THE OBJECT MAINTAINS ITS *STATE* (THE VALUES OF ITS INSTANCE VARIABLES), WHICH CAN CHANGE AND EVOLVE THROUGHOUT ITS LIFETIME.
- TO GET A NEWLY-CREATED OBJECT OFF TO A GOOD START, ITS NEWLY-ALLOCATED MEMORY MUST BE INITIALIZED TO A PROPER INITIAL STATE.
- AT THE BEGINNING OF AN OBJECT'S LIFE, THE JAVA VIRTUAL MACHINE (JVM) ALLOCATES ENOUGH MEMORY ON THE HEAP TO ACCOMMODATE THE OBJECT'S INSTANCE VARIABLES.

- 
- WHEN THAT MEMORY IS FIRST ALLOCATED, HOWEVER, THE DATA IT CONTAINS IS UNPREDICTABLE.
 - IF THE MEMORY WERE USED AS IS, THE BEHAVIOR OF THE OBJECT WOULD ALSO BE UNPREDICTABLE.
 - TO GUARD AGAINST SUCH A SCENARIO, JAVA MAKES CERTAIN THAT MEMORY IS INITIALIZED, AT LEAST TO PREDICTABLE DEFAULT VALUES, BEFORE IT IS USED BY ANY CODE.
 - THE JAVA LANGUAGE, HAS BUILT-IN MECHANISMS THAT HELP YOU ENSURE PROPER INITIALIZATION OF THE MEMORY OCCUPIED BY A NEWLY-CREATED OBJECT.
 - WITH PROPER USE OF THESE MECHANISMS, YOU CAN PREVENT AN OBJECT OF YOUR DESIGN FROM EVER BEING CREATED WITH AN INVALID INITIAL STATE.

JAVA LANGUAGE INITIALIZATION MECHANISMS

- **DEFAULT INITIAL VALUES:**

- IF YOU PROVIDE NO EXPLICIT INITIALIZATION TO INSTANCE VARIABLES, THEY WILL BE AWARDED PREDICTABLE *DEFAULT INITIAL VALUES*, WHICH ARE BASED ONLY ON THE TYPE OF THE VARIABLE
- THESE ARE THE DEFAULT INITIAL VALUES FOR BOTH INSTANCE AND CLASS VARIABLES.
 - BOOLEAN - FALSE, BYTE - (BYTE) 0, SHORT (SHORT) 0, INT 0, LONG 0L, CHAR \U0000, FLOAT - 0.0F
DOUBLE - 0.0D, OBJECT - REFERENCE NULL
- LOCAL VARIABLES ARE NOT GIVEN DEFAULT INITIAL VALUES. THEY MUST BE INITIALIZED EXPLICITLY BEFORE THEY ARE USED.

CONT ...

- **CONSTRUCTORS:**

- IN JAVA, CONSTRUCTORS ARE SIMILAR TO METHODS, BUT THEY ARE NOT METHODS.
- LIKE A METHOD, A CONSTRUCTOR HAS A SET OF PARAMETERS AND A BODY OF CODE.
- UNLIKE METHODS, HOWEVER, CONSTRUCTORS HAVE NO RETURN TYPE.
- LIKE METHODS, YOU CAN GIVE ACCESS SPECIFIERS TO CONSTRUCTORS, BUT UNLIKE METHODS, CONSTRUCTORS WITH PUBLIC, PROTECTED, OR PACKAGE ACCESS ARE NOT INHERITED BY SUBCLASSES.
- (ALSO, INSTEAD OF DETERMINING THE ABILITY TO INVOKE A METHOD, THE ACCESS LEVEL OF A CONSTRUCTOR DETERMINES THE ABILITY TO INSTANTIATE AN OBJECT.)
- AS WITH METHODS, YOU CAN OVERLOAD CONSTRUCTORS BY VARYING THE NUMBER, TYPES, AND ORDER OF PARAMETERS.

CONT ...

- **INSTANCE INITIALIZATION METHODS:**

- WHEN YOU COMPILE A CLASS, THE JAVA COMPILER CREATES AN INSTANCE INITIALIZATION METHOD FOR EACH CONSTRUCTOR YOU DECLARE IN THE SOURCE CODE OF THE CLASS.
- ALTHOUGH THE CONSTRUCTOR IS NOT A METHOD, THE INSTANCE INITIALIZATION METHOD IS.
- IT HAS A NAME, <INIT>, A RETURN TYPE, VOID, AND A SET OF PARAMETERS THAT MATCH THE PARAMETERS OF THE CONSTRUCTOR FROM WHICH IT WAS GENERATED.
 - **PUBLIC VOID <INIT>(COFFEECUP THIS) {...}**
- NOTE THAT <INIT> IS NOT A VALID JAVA METHOD NAME, SO YOU COULD NOT DEFINE A METHOD IN YOUR SOURCE FILE THAT ACCIDENTALLY CONFLICTED WITH AN INSTANCE INITIALIZATION METHOD.

CONT ...

- ALSO, THE **THIS** REFERENCE PASSED AS THE FIRST PARAMETER TO <INIT> IS INSERTED BY THE JAVA COMPILER INTO THE PARAMETER LIST OF EVERY INSTANCE METHOD.
- FOR EXAMPLE, THE METHOD **VOID ADD(INT AMOUNT)** IN THE SOURCE FILE FOR CLASS **COFFEECUP** WOULD BECOME THE **VOID ADD(COFFEECUP THIS, INT AMOUNT)** METHOD IN THE CLASS FILE.
- THE HIDDEN **THIS** REFERENCE IS THE WAY IN WHICH INSTANCE METHODS, INCLUDING INSTANCE INITIALIZATION METHODS, ARE ABLE TO ACCESS INSTANCE DATA.