1. Explain the differences between primitive and reference data types.
    - Primitive data types are basic data types that represent a single value. They are the building blocks of data manipulation in a program.
    - Reference data types, on the other hand, are data types that refer to an object. They do not hold the object's value directly, but rather hold a reference to the object's memory address.
    - primitive data types are stored on the stack, while reference data types are stored on the heap. This means that primitive data types are generally faster to access and manipulate than reference data types, but reference data types can store more complex data structures and can be more flexible to work with.

2. Define the scope of a variable (hint: local and global variable)
    - A local variable is a variable that is only accessible within the block of code in which it is defined.
    - A global variable, on the other hand, is a variable that is accessible from anywhere within the program.
3. Why is initialization of variables required?
    ○ It ensures that the variable has a known and defined value, rather than being undefined or having an unpredictable value.
    ○ It can help prevent errors and bugs in the program, as using an uninitialized variable can lead to unexpected behavior or results.
    ○ It can make the code more readable and self-explanatory, as the purpose and intended use of the variable are clear from the initialization.

4. Differentiate between static, instance and local variables.
    - A static variable is a variable that is associated with a class, rather than with a specific instance of that class. In other words, there is only one copy of a static variable that is shared among all instances of the class. Static variables are often used to store values that are common to all instances of a class, such as constants or counters.

    - An instance variable is a variable that is associated with a specific instance of a class. In other words, each instance of a class has its own copy of any instance variables defined by that class. Instance variables are often used to store data that is specific to a particular instance of a class, such as an object's state or properties.

    - A local variable is a variable that is defined within a block of code, such as a function or loop. Local variables are only accessible within the block of code in which they are defined, and they are often used to store temporary values that are needed only for the duration of that block of code.

5. Differentiate between widening and narrowing casting in java.
    - Casting refers to the process of converting a value of one data type to another data type. There are two main types of casting: widening casting and narrowing casting.

    - Widening casting, also known as upcasting, refers to the process of converting a value from a smaller data type to a larger data type. Widening casting is always safe, as the larger data type can hold all of the possible values of the smaller data type. In Java, widening casting can be done automatically (implicit casting) or explicitly (explicit casting).

6. The following table shows data type, its size, default value and the range. Filling in the missing values.

Type	Size	Default	Range
boolean	1 bit	false	true, false
byte	8 bits	0	-27 to +27-1
char	16 bits	\u0000	'\0000' to '\ffff'
short	16 bits	0	-215 to +215-1
int	32 bits	0	-231 to +231-1
long	64 bits	0	-263 to 263-1
float	32 bits	0.0	3.4E-38 to 3.4E+38
double	64 bits	0.0	-1.8E+308 to +1.8E+308

7. Explain the importance of using Java packages
    - Packages provide a way to specify access control for the classes and interfaces that they contain. For example, a package can be marked as "private," which means that the classes and interfaces within the package can only be accessed by other classes within the same package.

    - Packages provide a convenient way to reuse code. Classes and interfaces in a package can be easily shared and used in other Java programs by including the package in the program's classpath.

8. Explain three controls used when creating GUI applications in Java language.
    - Buttons: Buttons are used to trigger an action when clicked. Examples of buttons include JButton in the Swing library and Button in the AWT library.

    - Text fields: Text fields are used to input and display text. Examples of text fields include JTextField in the Swing library and TextField in the AWT library.

    - Menus: Menus are used to create a list of options that can be selected by the user. Examples of menus include JMenu and JMenuItem in the Swing library, and Menu and MenuItem in the AWT library.

9. Explain the difference between containers and components as used in Java.
    containers and components are used to create the user interface of a GUI application. Containers are used to hold and organize other GUI components, while components are the individual elements of the user interface such as buttons, text fields, and lists.

10. Write a Java program to reverse an array having five items of type int.

```
public class ArrayReverse {
  public static void main(String[] args) {
    // Declare and initialize the array
    int[] array = {1, 2, 3, 4, 5};

    // Print the original array
    System.out.println("Original array: " + Arrays.toString(array));

    // Reverse the array
    reverseArray(array);

    // Print the reversed array
    System.out.println("Reversed array: " + Arrays.toString(array));
  }

  // Method to reverse an array
  public static void reverseArray(int[] array) {
    // Loop through the array, swapping the elements at the front and back
    for (int i = 0; i < array.length / 2; i++) {
      int temp = array[i];
      array[i] = array[array.length - 1 - i];
      array[array.length - 1 - i] = temp;
    }
  }
}
```

11. Programs written for a graphical user interface have to deal with "events."
    Explain what is meant by the term event.
    Give at least two different examples of events, and discuss how a program might respond to those events.

    - An event in the context of a graphical user interface (GUI) is an action or occurrence that the program can respond to. Events can be triggered by the user, the system, or other external factors.

    - Here are two examples of events that might occur in a GUI program:

    - User input events: These events are triggered by the user interacting with the program, such as clicking a button, typing in a text field, or moving the mouse. A program might respond to user input events by performing an action, such as displaying a message, updating the user interface, or making a request to a server.

    - System events: These events are triggered by the system or the operating environment, such as the program starting up, the window being resized, or the system going to sleep. A program might respond to system events by updating the user interface or changing the way it operates.

    - In general, a program responds to events by executing a piece of code, known as an event handler, that is associated with the event. The event handler is responsible for performing the appropriate action in response to the event. In Java, events are typically handled using event listeners, which are objects that listen for events and invoke the appropriate event handler when an event occurs.

12. Explain the difference between the following terms as used in Java programming.
    Polymorphism and encapsulation
    Method overloading and method overriding
    Class and interface
    Inheritance and polymorphism

Polymorphism and encapsulation:
Polymorphism refers to the ability of a single entity (such as a method or object) to take on multiple forms. In Java, polymorphism can be achieved through inheritance, interfaces, and method overloading/overriding.
Encapsulation refers to the idea of wrapping data and methods that operate on that data within a single unit, or object. Encapsulation helps to protect the data from outside access and modification, and is achieved through the use of access modifiers and class constructors.

Method overloading and method overriding:
Method overloading refers to the ability to have multiple methods with the same name, but with different parameter lists. Method overloading allows for the creation of methods that perform similar tasks, but with different input.
Method overriding refers to the ability to have a method in a subclass with the same name and parameter list as a method in the superclass. Method overriding allows for the subclass to provide a specific implementation of the method, rather than using the implementation from the superclass.

Class and interface:
A class is a template for creating objects that defines the properties and behaviors of those objects. A class can contain fields, methods, and constructors.
An interface is a template for classes that defines a set of methods that the implementing class must have. An interface does not contain any implementation code, but rather just the method signatures.

Inheritance and polymorphism:
Inheritance refers to the ability of a class to inherit properties and behaviors from a superclass. Inheritance allows for the creation of a class hierarchy, where subclass can inherit and extend the functionality of a superclass.
Polymorphism refers to the ability of a single entity (such as a method or object) to take on multiple forms. Polymorphism can be achieved through inheritance, interfaces, and method overloading/overriding. In the context of inheritance, polymorphism allows for a subclass to be treated as if it were the same type as its superclass, even though it may have additional or different behavior.

13. Using examples, explain the two possible ways of implementing polymorphism.
Show your code in java.

Polymorphism refers to the ability of a single entity (such as a method or object) to take on multiple forms. There are two main ways to implement polymorphism in Java: through inheritance and through interfaces.

Examples:

Polymorphism through inheritance:

```
// Base class
class Animal {
  public void makeNoise() {
    System.out.println("Some generic animal noise");
  }
}

// Subclass of Animal
class Dog extends Animal {
  public void makeNoise() {
    System.out.println("Bark!");
  }
}

// Subclass of Animal
class Cat extends Animal {
  public void makeNoise() {
    System.out.println("Meow!");
  }
}

public class Main {
  public static void main(String[] args) {
    // Create an array of Animals
    Animal[] animals = {new Dog(), new Cat()};

    // Call the makeNoise method on each animal
    for (Animal animal : animals) {
      animal.makeNoise();
    }
  }
}
```

In this example, the Animal class has a makeNoise method that is overridden in the Dog and Cat subclasses. When the makeNoise method is called on an instance of Dog or Cat, the subclass's implementation of the method is used, rather than the one from the superclass. This allows each subclass to have its own specific behavior, while still being treated as an instance of the Animal class.