Why char array are preferred over Strings to store passwords?

1. With plain String you have much higher chances of accidentally printing the password to logs, monitors or some other insecure place. char[] is less vulnerable.

Consider this:

public static void main(String[] args) {

String pw = "Password";

System.out.println("String: " + pw);

pw = "Password".toCharArray();

System.out.println("Array: " + pw);

}

Prints:

String: Password

Array: [C@5829428e

1. It would seem logical to collect and store the password in an object of type java.lang.String. However, here's the caveat: Objects of type String are immutable, i.e., there are no methods defined that allow you to change (overwrite) or zero out the contents of a String after usage.   
     
   Since, we cannot overwrite the value, it remains in the intern pool. Anybody who has access to the intern pool can access the string contents.  
     
   This feature makes String objects unsuitable for storing security sensitive information such as user passwords. You should always collect and store security sensitive information in a char array instead.

Why Strings are immutable in Java?

1. Choice by Language designers. You have C++ where Strings are mutable.
2. String pool exists because Strings are immutable. In turn, it enhances the performance by saving heap memory.
3. Thread safe, as we cannot make changes to String object and hence, it cannot be changed when accessed by multiple threads.

Why Generics are used: to make Data Strucutres possible

Basics of Generics

What happens when we don’t define a generic type while creating objects.

Adding bounds to generic type

Invariance

Covariance - accept subtypes

Contravariance - accept supertypes

Array<X> -> add anything that extends from X -> Covariant

Array<X> -> add anything that X is extending from -> Supertypes of X -> Contravariance

Some more points:

1. Java is pass by value or pass by reference: Java supports only pass by value. When primitives are passed to parameter variables, then the changes are not reflected to calling function, as it only creates new copies of variables.   
   When non primitives are passed to parameters, then copies of variables is created and data of variable stores reference to actual object stored in heap. Immutable objects don’t reflect back the changes, but mutable objects reflect back the changes to calling function.  
   Refer this link: <https://www.scaler.com/topics/pass-by-value-and-call-by-reference-in-java/>
2. What problem does generics solve in Java programming?
   1. Generics make data structures possible
   2. Type safety
   3. We can have same class to be used with any data type
3. Generics are invariant by default, covariant accepts sub types, contravariant accepts super types, invariance is none of these  
     
   We can create bounded types to remove invariance in generics. Bounded types gives us covariance.
4. Where would we need contravariance.
5. LinkedList implements both Queue and List intrerface.
6. Difference between iterable and iterator